Diplomarbeit

GRAPH-BASED MOTION SEGMENTATION OF OPTICAL FLOW

Ausgeführt zum Zwecke der Erlangung des akademischen Grades eines Diplom-Ingenieurs

unter der Leitung von

Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Markus Vincze

und

Dipl.-Ing. Johann Prankl

Institut für Automatisierungs- und Regelungstechnik (ACIN)

eingereicht an der Technischen Universität Wien

Fakultät für Elektrotechnik und Informationstechnik

von

Barbara NEUHERZ

Matr.Nr.: 9925936

Ortliebgasse 33/18, 1170 Wien

Wien, im Juni 2009

Abstract

In this project the usage of optical flow for motion segmentation was evaluated. The optical flow is represented by a vector field that describes the displacement of pixels in subsequent frames. The motion vectors hold the magnitude and direction corresponding to the movement of the pixels. They can be either computed for every pixel in the image or just for certain, important pixels, such as corner points. The segmentation process clusters pixels with similar motion vectors to regions that can be used for further processing, such as tracking of objects or collision avoidance for home-robotics.

The computation of the optical flow for every pixel in the image (dense optical flow) is computationally extensive. Although modern graphic hardware offers the possibility of real-time computation an attractive alternative to save resources is the computation of the motion vectors only for certain points in the image (which is called sparse optical flow).

This work covers the theoretic fundamentals as well as a detailed description of the implemented segmentation algorithm. Different image sequences produced in the lab and datasets from an evaluation-database were taken to demonstrate the segmentation of dense optical flow and sparse optical flow, respectively. Additionally, both methods are compared. From the results it is concluded that the less computationally extensive sparse optical flow offers a comparable performance and is therefore preferable over the dense optical flow.

Kurzfassung

Im Rahmen dieses Projects wurde untersucht, inwieweit sich Optischer Fluss zur Bewegungssegmentierung eignet. Der Optische Fluss ist ein Vektorfeld, das die Bewegung jedes Pixels eines Bildpaares beschreibt. Die Bewegungsvektoren, die mit ihrer Amplitude und Richtung diese Positionsänderung festhalten, können für jedes Pixel im Bild oder nur für einzelne, wichtige Pixel, z.B. Eckpunkte, berechnet werden. Im Zuge der Segmentierung werden Pixel mit ähnlichen Bewegungsvektoren zu einheitlichen Regionen zusammengefasst, die in weiteren Verarbeitungsschritten z.B. zur Verfolg von Objekten oder zur Kollisionsvermeidung im Bereich der Heim-Robotik verwendet werden können.

Die Berechnung vom Optischen Fluss für jedes Pixel im Bild (man nennt diesen *dense optical flow*) ist sehr rechenintensiv, wenngleich moderne Grafik-Hardware eine Berechnung in Echtzeit möglich macht. Die Alternative, diesen nur für ausgewählte Bildpunkte zu berechnen (dieser wird *sparse optical flow* genannt), erscheint daher besonders attraktiv.

In der vorliegenden Arbeit werden zunächst die theoretischen Grundlagen erläutert sowie der implementierte Algorithmus für die Segmentierung genau erklärt. Anhand von im Labor aufgenommenen und aus einer Evaluierungs-Datenbank stammenden Bildsequenzen wird sowohl die Segmentierung von *dense optical flow*, als auch von *sparse optical flow* demonstriert und ein Vergleich beider Methoden gezogen. Es zeigt sich, dass der weniger rechenintensive *sparse optical flow* oftmals ähnlich gute Resultate liefert und damit aufgrund seiner Effizienz vorzuziehen ist.

Acknowledgments

First and foremost, I would like to thank Prof. Dr. Markus Vincze for the realization of this project. Special thanks goes to my supervisor DI Johann Prankl for this support and constructive conversations during the course of the project. It was a great pleasure to experience the relaxed atmosphere within the Vision for Robotics (V4R) group.

Furthermore, I owe many thanks to my mother and my brothers Markus and Wolfgang for getting me involved with the field of electrical engineering. I am grateful to my sister in law Manuela and to Christoph Natorski who enrich my life to a great extend.

I would like to express my greatest gratitude to my partner Stefan Kostner. I appreciate his support and guidance throughout many years.

I wish to credit my close friend Antitza Dantcheva. Over the years she has become an important person in my life. I enjoyed our numerous conversations very much.

I take the opportunity to thank my colleagues of the working group *Electronics 1* at the Institute of High Energy Physics of the Austrian Academy of Sciences for the friendly environment they have provided me during the past six years. My special thanks goes to the group leader DI Anton Taurok for his patience and for motivating me to carry on with my studies.

Finally, I express my acknowledgment to the PhD students at the Industrial Sensor Systems Group at ISAS/TU-Vienna for their relaxing attitude at our get-togethers.

Barbara Neuherz

Nomenclature

$d(i,j)$	difference image
$d_{cum}(i,j)$	cumulative difference image
i, j or x, y	pixel coordinates at time t
$f(x,y,t)$	image frame
dx, dy	changes in position during time dt
f_x, f_y, f_t	partial derivations of the frame $f(x,y,t)$
u, v	velocity in x and y direction
$\mathbf{c}$	motion vector
$grad(f)$	2D image gradient
$E^2(x,y)$	squared error quantity
u_x^2, u_y^2, v_x^2, v_y^2	squared partial derivatives of the velocity in x and y direction
λ	Lagrange multiplier
$\bar{u}$, $\bar{v}$	mean values of the velocity in x and y direction
H	homogeneity criterion
$v_i \in V$	vertex; pixel in an image
$(v_i, v_j) \in E$	edges; connection between two vertices
$w((v_i, v_j))$	edge weight dealing with the magnitude of $\mathbf{c}$; measures dissimilarity between vertices connected by that edge
$w_{angle}((v_i, v_j))$	second edge weight dealing with the angle of $\mathbf{c}$; measures dissimilarity between vertices connected by that edge
$G = (V, E)$	connected undirected graph; formed by vertices and edges
$cut(A,B)$	minimum cut of two disjoint regions A and B
$\overline{c}(A,B)$	minimum mean cut considers the boundary length
$Ncut(A,B)$	normalized cut measures dissimilarity between and similarity within regions
$\tau(C)$	threshold function; used by the segmentation algorithm
k	constant parameter used by $\tau(C)$; specified by the user
$IntDif(C)$	internal difference; largest weight within a component
$MIntDif(C_i, C_j)$	minimum internal difference between two components
feature space	consists of adjacent pixels that can be mapped to non-neighboring pixels in the real image
c_{angle}	constant parameter used by the cluster condition; specified by the user

Contents

Chapter 1

Introduction

While a single image contains static information about an entire scene, subsequent images from a movie (at least two frames) additionally contain dynamic information which represents the motion of objects. Motion in general is a robust information that can be easily extracted from two subsequent frames. It is represented by the optical flow which is a vector field that describes the displacement of pixels. The motion vectors hold the magnitude and direction corresponding to the movement of the pixels. They can be either computed for every pixel in the image (dense optical flow) or just for certain, important pixels, such as corner points (sparse optical flow).

To extract information about moving objects (as opposed to moving pixels) regions of similar motion vectors are clustered. This process is called motion segmentation. Non-moving objects for which the motion vector is zero are clustered together and form one background region, while each well defined moving object is represented by its own region. Compared to an image that contains thousands of pixels, a segmentation output contains only a few regions. The focusing on the interesting (moving) objects comes along with a dramatic reduction of data.

In the real world environment the detection of moving objects is a very important task, because serious danger from collisions can be avoided. A comprehensive example is given in [Twe09]: if a pedestrian crosses a street, the non-moving vehicles are of less interest, because the threatening danger originates from the moving ones. Hence the pedestrian's brain will segment the motions in the scene by focusing on the moving vehicles and *clustering* the static ones with the background. In this example, as well as in many applications the knowledge that something is moving is of higher priority, than the knowledge about what is moving (in this example a car or a bus). The wide range of applications, that are based on motion segmentation, includes for example object detection [VJS05], object tracking [SGK00], surveillance [HHD00], and collision avoidance for robots [OAT+07].

In this work the segmentation of sparse and dense optical flow images is evaluated. Image sequences from an evaluation database as well as images from the lab environment are used as inputs to test the performance of the segmentation. The results are compared and discussed and possible improvements are highlighted.

A brief review on optical flow and motion segmentation

As outlined in [ZPB07], Horn and Schunck [HS80] first proposed the fundamental work concerning the computation of optical flow from an image sequence in 1980. In the following years based on their model further improvements were published in [NE86], [WB02], [BA93], [ADK99],

[PBB+06], and [ZPB07]. In general those methods are used to create dense optical flow images. Its alternative is the sparse optical flow, that is resource-saving and robust under noise, because motion vectors are only computed for certain pixels in the image that have high information content (corner points). Here, the extraction of the corner points plays a major role. The Kanade-Lucas-Tomasi (KLT) feature tracker is an established corner tracking algorithm. Its basic idea was first published in 1981 in the early work of Lucas and Kanade [LK81] and further improved in [TK91] and [Bou00].

In [ZLS08] the authors show that motion segmentation algorithms can be classified according to their main attributes, e.g., *feature-* and *dense-based*, *oclussion*, *multiple object* and *missing data* handling, *robustness* under noise, used *camera model*, requirements concerning *prior knowledge* or *training* sequences, *temporary stopping* of objects. Grouping some of those attributes allows following categorization:

- *Image difference* is an old and simple technique that is based on the dense-representation of objects. A binary image contains the calculated difference of two subsequent grey-scale frames (1 for changing pixel, 0 for static pixel). This method is sensitive to noise and to light changes. It can handle *occlusion* and *multiple objects*, but it has difficulties with *moving cameras* and *temporary stopping* objects. Detailed information can be found in [CSE05], [CC06], [LYY07], [CFM07].

- *Statistical approaches* are also based on object's dense-representation with the aim to assign pixels to either the foreground or the background. These approaches can handle *multiple objects*, *occlusion* and *temporary stopping* objects and is known to be *robust*. Additionally, *prior knowledge* is often required. Following classification can be made:

  - *Maximum A posteriori Probability* (MAP) is based on Bayes rule and is often used with other techniques, see [RH01], [CS05], [SZHL07].
  - *Particle Filter* (PF) constructs a sample-based probability density function to track the variable's changes over time, see [VTY07].
  - *Expectation Maximization* (EM) is a iterative method with guaranteed convergence. Observed data is represented by a parameter model where the parameters are estimated by computing the Maximum Likelihood function, see [Bor04] and [SGHG08].

- *Optical flow* is well suited for motion segmentation [ZSWL07]. As mentioned above it is described by a vector field that represents the displacement of pixels and was first published in 1980 [HS80]. Optical flow has problems with *temporary stopping* objects and *occlusion*. Additional methods are necessary to eliminate this drawbacks. Beside this, the optical flow is sensitive to noise and light changes. Nowadays, its high computational resource demand is a minor problem because optimized graphic processing units can be used.

- *Wavelets* provide a strong mathematical framework for analyzing functions at various scales. It is a powerful tool in image processing[1] (see [Wis97] and [KLGW98]). The multi-resolution (one image at various resolutions) is used to characterize the structure of an image. For example, information about different depth-planes can be extracted that is used to solve the *occlusion* problem.

[1] Imagine a continuous-valued brightness function of an one-dimensional image. The idea of wavelet transformation is the approximation of this function using a discrete set of values.

- *Layers* are used to assign objects in an image to different planes. In 3D, the layers represent depth-layers, where objects depending on their depth are assigned to. In 2D, objects with similar velocity and direction are assigned to the same layer. More information can be found in [KTZ08] or [BVZ99]. Layers are useful to solve the *occlusion* problem like human beings. Unfortunately, the algorithm itself is numerically extensive.

- *Factorization* was first introduced in 1992 by Tomasi and Kanade [TK92]. The idea is to use features, tracked over several frames, to recover structure and motion. For those tracked features a trajectory matrix **W** can be defined, that contains the position of the tracked features over several frames. This matrix can be separated (factorized) into a matrix **M**, that holds the motion information, and a matrix **S**, that contains the structure information. This method has no problem with *temporary stopping objects*, because features can be extracted in any case. Since 1992 many approaches improved and extended the basic idea, see [CK98], [YP06], [JSL+07], etc.

Once the motion segmentation is done, additional post-processes have to be executed that interpret or re-work the segmentation output. For a person it is easy to analyze the content of a scene and to find decisions. For a machine this is still an unsolved problem. *Artificial intelligence* (AI) is a branch of computer sciences, that studies perceptual functions (e.g., decision finding) to implement them on machines. An approach to define intelligence in general and artificial intelligence in particular is given in [HB04, p. 12], a proposal for combination of computer vision research from robotics and artificial intelligence is presented in [SPV09].

Chapter 2

Theory

In this chapter basic knowledge is transferred to the user. Chapter 2.1 gives an introduction to motion analysis methods, such as difference image or optical flow, the latter used in this project. An overview about segmentation in general is given in chapter 2.2 and about region-based segmentation algorithms in detail in 2.3. The Delaunay triangulation method, its fundamental idea, and construction is highlighted in 2.4.

2.1 Motion analysis

In recent years, computational power increased and new applications for motion analysis emerged. Some robotic applications, e.g., robot navigation, are based on real-time processing. In contrast, obtaining 3D shape and relative depth from motion is often done off-line. Prior knowledge like time interval between consequent frames or camera motion, decreases the complexity of the analysis. Because the motion analysis technique depends on the available information no general algorithm exist.

Three main groups of motion-related problems can be defined [SHB99, p. 679-680]:

- Motion detection,
- moving object detection and
- derivation of 3D object properties.

Motion detection is the simplest problem. It uses a static camera that records any detection. Thus it is often used for security purposes.

The *moving object detection* is more difficult compared to *motion detection* and is most complex if both camera and object move. The detection of moving objects is based on motion-based segmentation algorithms. It is also possible to detect not the moving object itself but the trajectory of its motion or to predict its future location.

A comprehensive summary of approaches dealing with the *derivation of 3D object properties* from a set of two dimensional projections can be found in [SHB99, p. 680].

The difference image and the optical flow are two common methods to detect movements in an image and are described in the following.

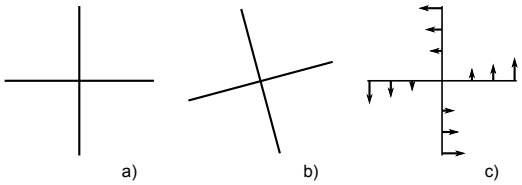

Figure 2.1: a) Artificial image at time t. b) Image at time $t + dt$. c) Corresponding optical flow and computed motion vectors (marked for certain pixels).

2.1.1 Difference image

The calculated difference between two subsequent grey-scale frames can be used to create the difference image $d(i, j)$ (see [SHB99, p. 682]) that is a binary image where zero values represent non-moving pixels and non-zero values represent moving pixels,

$$d(i,j) = \begin{cases} 0 & \text{if } |f_1(i,j) - f_2(i,j)| < \epsilon \\ 1 & \text{otherwise,} \end{cases} \tag{2.1}$$

where $f_1(i, j)$ and $f_2(i, j)$ are the subsequent frames, i and j are the pixel coordinates and ϵ is a threshold to ignore very small movements caused by noise or inaccuracies.

The difference image contains the information if a pixel position changed[1] between the subsequent frames but it does not contain the direction of the motion. Therefore the cumulative difference image $d_{cum}(i, j)$ can be constructed from n subsequent frames ([SHB99, p. 683]) with the first frame as reference image,

$$d_{cum}(i,j) = \sum_{k=1}^{n} a_k |f_1(i,j) - f_k(i,j)|, \tag{2.2}$$

where $f_k(i, j)$ are the subsequent frames and a_k is a factor to weight the frames.

This method has two major disadvantages [SHB99, p. 684]:

- For example, if a rectangular object without pattern moves horizontal and parallel to its object boundary motion of only the left and right hand side are detected. The same is true for an object that moves vertical. In this case motion of the upper and bottom side are detected.
- If only a part of a boundary of the first frame is visible in the second frame the motion can not be determined correctly. This is called aperture problem.

2.1.2 Optical flow

The optical flow is represented by a vector field that describes the displacement of pixels in subsequent frames (within a defined time interval dt), see [SHB99, p. 685]. The motion vectors hold the magnitude and direction corresponding to the movement of the pixels. Fig. 2.1 shows an artificial example of two subsequent images and its corresponding optical flow[2].

[1] Non-zero values represent pixels with motion.

[2] In this example the motion vectors are computed for certain pixels in the image thus it is called sparse optical flow.

Optical flow computation

The optical flow computation is based on following assumptions ([SHB99, p. 686]):

- The brightness of objects is constant and
- neighboring pixels move in similar manner.

A continuous, grey-scale image $f(x, y, t)$ can be expressed as Taylor series,

$$f(x + dx, y + dy, t + dt) = f(x, y, t) + f_x dx + f_y dy + f_t dt + O(\delta^2), \tag{2.3}$$

where x and y are the pixel coordinates at time t, dx and dy are the changes of position during time dt, f_x, f_y and f_t are the partial derivations of the frame $f(x, y, t)$ and $O(\delta^2)$ is a place holder for the higher order terms.

The location of a moving pixel in the two subsequent frames can be written as

$$f(x + dx, y + dy, t + dt) = f(x, y, t). \tag{2.4}$$

If dx, dy and dt are very small the higher order terms can be ignored and eq. 2.3 can be rearranged to

$$-f_t = f_x \frac{dx}{dt} + f_y \frac{dy}{dt} = f_x u + f_y v = grad(f)\mathbf{c}, \tag{2.5}$$

see [SHB99, p. 687], where u and v are the velocity in x and y direction that can be summarized as motion vector $\mathbf{c}$ and $grad(f)$ is the two dimensional image gradient.

To solve eq. 2.5, which is an equation in two unknowns, a further constraint, the so called *smoothness constraint*, has to be defined. The detailed approach can be found in [HS80] but it turns out to minimize following formula:

$$E^2(x, y) = (f_x u + f_y v + f_t)^2 + \lambda({u_x}^2 + {u_y}^2 + {v_x}^2 + {v_y}^2), \tag{2.6}$$

where $E^2(x, y)$ is the squared error quantity, u_x^2, u_y^2, v_x^2 and v_y^2 are the squared partial derivatives and λ is a Lagrange multiplier. The first term represents eq. 2.5 and the second term represents the smoothness criterion. Minimizing eq. 2.6 yields to two differential equations

$$(\lambda^2 + f_x^2)u + f_x f_y v = \lambda^2 \bar{u} - f_x f_t \text{ and} \tag{2.7}$$

$$f_x f_y u + (\lambda^2 + f_y^2)v = \lambda^2 \bar{v} - f_y f_t, \tag{2.8}$$

where $\bar{u}$ and $\bar{v}$ are the mean values of the velocity in x and y direction. A solution for the differential equations eq. 2.7 and 2.8 is

$$u = \bar{u} - f_x \frac{P}{D} \text{ and} \tag{2.9}$$

$$v = \bar{v} - f_y \frac{P}{D} \text{ and} \tag{2.10}$$

where

$$P = f_x \bar{u} + f_y \bar{v} \text{ and } D = \lambda^2 + {f_x}^2 + {f_y}^2. \tag{2.11}$$

The determination of the optical flow is based on Gauss-Seidel iteration method.

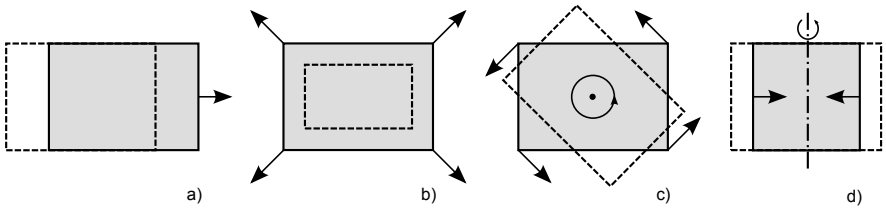

Figure 2.2: a) Translation at constant distance. b) Translation in depth. c) Rotation at constant distance. d) Rotation perpendicular to the view axis. [SHB99, p. 693, fig. 15.9]

If dx, dy and dt are not small[3] the higher order terms in eq. 2.3 can not be ignored. At least the second order terms have to be considered in the Taylor series and makes the optical flow computation rather complex. See [SHB99, p. 689] for more details.

Very important for the optical flow computation is the contrast. For example if a smooth sphere rotates under constant illumination the optical flow is zero because the change is not noticed [SHB99, p. 693].

The optical flow can be either computed for every pixel in the image or just for certain, important pixels, such as corner points. The former is called *dense optical flow*, the latter *sparse optical flow*.

In this project for the computation of the dense optical flow used for further processing the Win32 library from [ZPB07] was used.

Optical flow in motion analysis

Optical flow is used to study the use cases:

- Moving object and static camera,
- static object and moving camera and
- moving object and camera.

Motion can be a combination of translation and rotation (compare [SHB99, p. 693]). Depending on the movement the following basic motion vectors can be defined:

- Translation at constant distance from the camera: parallel motion vectors.
- Translation in depth relative to the camera: motion vectors with common focus of expansion (FOE)[4].
- Rotation at constant distance about the view axis: concentric motion vectors.
- Rotation perpendicular to the view axis: motion vectors starting from straight line segment.

An illustration of the above statements are shown in fig. 2.2.

Compared to the difference image, see chapter 2.1.1, the optical flow computation has the advantage that each optical flow field calculated from two subsequent images contains motion vectors with defined magnitude and direction. Unfortunately it holds the same disadvantages like the difference image.

[3] This is the case if the time interval between subsequent images is not small enough.

[4] For the translation at constant distance from the camera the common focus of expansion is at infinity.

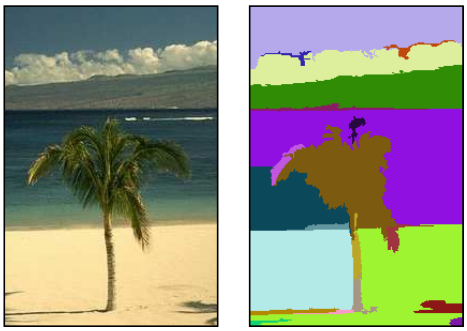

Figure 2.3: a) Color input image to be segmented. b) Complete segmentation by clustering pixels with similar color value.

Figure 2.4: a) Color input image at time t. b) Color input image at time $t+dt$. c) Partial segmentation by clustering pixels with similar motion vector hence the moving objects (ball in the upper left, three persons in the middle) result while non-moving objects (background, ground) are clustered together.

2.2 Segmentation

Segmentation is a process at which an image is split into meaningful areas (such as areas of similar color) to reduce and simplify the information content. The basic requirement for segmentation can be expressed as

$$R = \bigcup_{i=1}^{S} R_i \qquad R_i \cap R_j = 0 \qquad i \neq j, \tag{2.12}$$

where S is the total number of segmented regions R_i, see [SHB99, p. 124].

Since the smallest entity of an image is a pixel, segmentation means the clustering of pixels with similar characteristics to form one region. There is a difference between *complete* and *partial* segmentation. While *complete segmentation* means that each object in the image has its corresponding region in the segmented output, see fig. 2.3[5], *partial segmentation* reduces information such that different objects in the image can be merged to one region in the output, see fig. 2.4[6].

Important cues for segmentation are the pixel characteristics, e.g., brightness, color, and motion. Furthermore the important property of the surface of objects is its homogeneity. Ho-

[5] Images were taken from [FH04].

[6] Color input images were taken from the optical flow evaluation database, see `http://vision.middlebury.edu/flow/data/`.

mogeneity and pixel characteristic together provide a reliable clustering algorithm.

The main problem of segmentation is the ambiguity of information in the image caused by noise or aperture problems. To avoid this the image quality should be as high as possible. This can be achieved by taking images at constant illumination with high-quality cameras.

Segmentation algorithms can be divided into three groups that depend on the features they use:

- Global knowledge,
- edge-based and
- region-based.

If *global knowledge* about the objects in the image is available a simple thresholding method[7] can be used to segment the image. This method suits best for images where objects have to be separated from the background.

Edge-based and *region-based* methods use the pixel characteristics. Hence every region has its own boundary and every closed boundary describes a region the results of the segmentation may differ. A good result yields a combination of both methods [SHB99, p. 123]. The *region-based* methods, used in this work, are explained more detailed in following section.

2.3 Region-based segmentation algorithms

Region-based segmentation methods are best suited for noisy images where detection of borders is very difficult and thus edge-based segmentation methods yield bad results [SHB99, p. 176].

The important property of surfaces of objects is their homogeneity. A homogeneity criterion H can be defined such that every segmented region has to satisfy eq. 2.12 as well as following conditions:

$$H(R_i) = TRUE \qquad i = 1, 2, ..., S \text{ and} \tag{2.13}$$

$$H(R_i \cup R_j) = FALSE \qquad i \neq j \qquad R_i \text{ adjacent to } R_j, \tag{2.14}$$

where $H(R_i)$ is the homogeneity evaluation of region R_i, see [SHB99, p. 177]. The homogeneity criterion can use for example an average grey-level, color or texture properties, and direction and magnitude of motion vectors.

Methods discussed in the following deal with images in two dimensions and can be extended to three dimensions of course. The regions obtained in three dimensions are volumes while in two dimensions are planar.

2.3.1 Region merging and region splitting

Region Merging

This segmentation approach is applied to the raw image data. Every pixel represents its own region[8] and has its own region description, that can be computed. During the merging process the region descriptions of adjacent regions are compared. If they match both regions are joined

[7] E.g., grey-level thresholding.

[8] This regions fulfill eq. 2.13 but not probably eq. 2.14.

to form a new one otherwise the process continues with the next adjacent region [SHB99, p. 178].

This *merging condition* is very rigorous and should be lightened such that the region descriptions have to match within a certain range $\pm\Delta x$,

$$f(x_i - \Delta x) \leq f(x_j) \leq f(x_i + \Delta x), \tag{2.15}$$

where $f(x_i)$ and $f(x_j)$ are the region descriptions of region R_i and R_j respectively and $f(x_i - \Delta x)$ and $f(x_i + \Delta x)$ is the lower and the higher threshold, respectively.

Equivalent to the comparison of region descriptions is the definition of edges between adjacent regions and a weight assigned to the edge[9]. Using this weight and a preset threshold an edge is going to be determined as weak or strong,

$$v_{ij} = \begin{cases} 0 & \text{if } w_{ij} = |f(x_i) - f(x_j)| < T \\ 1 & \text{otherwise} \end{cases} \tag{2.16}$$

where $v_{ij} = 0$ and $v_{ij} = 1$ indicates a weak[10] and a strong edge respectively and T the certain threshold [SHB99, p. 179]. The merging condition then is formulated such that pixels of weak edges, $v_{ij} = 0$, are merged together.

Region splitting

In case of region splitting the whole image is defined as one region[11] and is subsequently split into sub-regions until eq. 2.13 is satisfied.

This approach seems to be dual to region merging but like in case of edge-based and region-based segmentation methods the results here differ even if the homogeneity criterion is the same.

Region splitting and merging

This approach combines the advantages of region splitting and region merging and defines pyramid levels. First the whole image is defined as one region with a pyramid of the same size. This region does not satisfy eq. 2.13 hence it is split into four child-regions and a new pyramid of one-fourths size is defined. If any of the four child-regions does not satisfy eq. 2.13 they are split into four child-regions again and so on. Fig. 2.5 demonstrates this with an artificial image.

The splitting process can be continued as long as either a segmentation with acceptable error or a certain resolution is achieved. If the splitting process is stopped the merging process starts and joins all adjacent regions with similar values of homogeneity [SHB99, p. 182].

2.3.2 Graph-based segmentation

Graph-base segmentation is an efficient[12] clustering algorithm. It needs to define vertices $v_i \in V$ which corresponds to the pixel in the image and edges $(v_i, v_j) \in E$ between vertices. A weight $w((v_i, v_j))$ can be assigned to each edge which measures the similarity between neighboring pixels. Vertices and edges together form the connected, undirected graph $G = (V, E)$. The aim

[9] This definition is used by the graph-based segmentation, see sec. 2.3.2.
[10] Weak edges are also called *crack edges*.
[11] This region does not fulfill eq. 2.13
[12] In respect of computing time and achievable success.

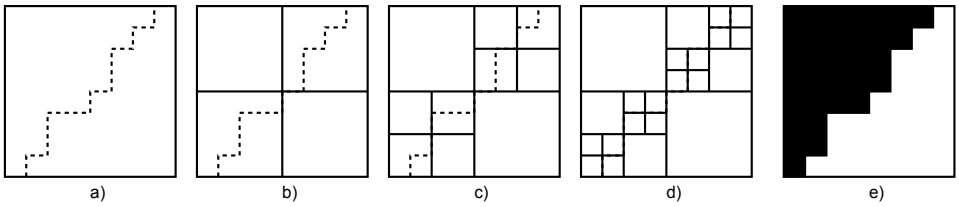

Figure 2.5: a) Artificial image to be segmented with a pyramid of the same size. b-d) Splitting of regions if eq. 2.13 is not fulfilled with smaller pyramids. e) Segmented output after merging adjacent regions with similar characteristics.

is to build a graph such that vertices which belong together are connected. A set of vertices provides a lot of possibilities how to connect them. Hence a constraint has to be expressed to lower the possibilities and to provide only one graph at best. The constraint can take the edge weights into account and is expressed as a cost function that has to be minimized. Examples for cost functions are the *minimum cut*, the *minimum mean cut*, and the *normalized cut*.

Minimum cut

For the minimum cut $cut(A, B)$ the sum of the edge weights between two disjoint regions has to be a minimum. Its cost function can be expressed as

$$cut(A, B) = \sum_{v_i \in A, v_j \in B} w((v_i, v_j)), \tag{2.17}$$

where A and B are two disjoint regions ($A \cup B = V$) and $w((v_i, v_j))$ is the edge weight between the vertices v_i and v_j.

Minimum mean cut

The cost function of the minimum mean cut $\overline{c}(A, B)$ extends the minimum cut by considering the boundary length ([DW01]). It can be expressed as

$$\overline{c}(A, B) = \frac{cut(A, B|w((v_i, v_j)))}{cut(A, B|l)}, \tag{2.18}$$

where A and B are two disjoint regions ($A \cup B = V$), $cut(A, B|w((v_i, v_j)))$ is the cost function for the minimum cut (see eq. 2.17) and $cut(A, B|l)$ is the cut boundary length.

Normalized cut

The normalized cut criterion $Ncut(A, B)$ explained in [SM00] measures the total dissimilarity between regions as well as the total similarity within regions and puts them into relation. The cost function is expressed as

$$Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)}, \tag{2.19}$$

where $assoc(A, V) = \sum_{v_i \in A, v_j \in V} w((v_i, v_j))$ is the total connection between vertices inside and outside region A, $assoc(B, V)$ is defined similar and $cut(A, B)$ is the minimum cut (see eq. 2.17).

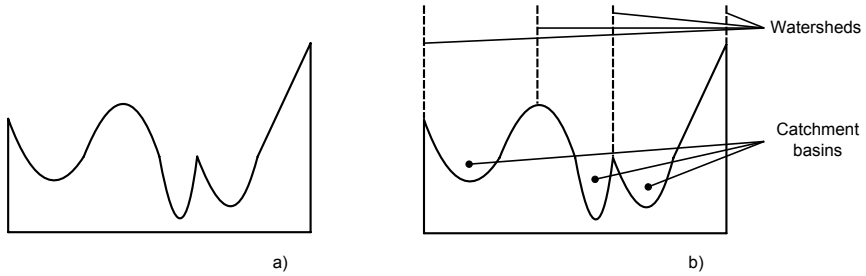

Figure 2.6: a) One-dimensional grey-level relief. b) Region edges correspond to high watersheds and region compounds correspond to catchment basins. Figure based on [SHB99, p. 187].

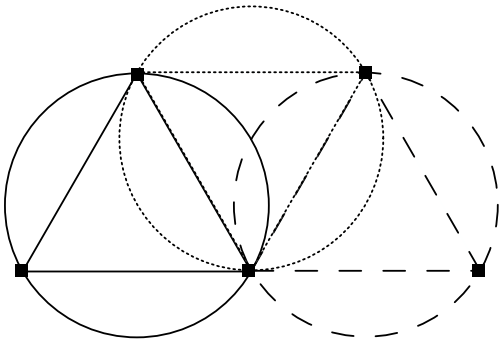

Figure 2.7: The Delaunay triangulation in a two-dimensional space with empty circumcircles.

2.3.3 Watershed segmentation

Any grey-level image can be considered as topographical surface where the grey-level value of the pixels is interpreted as their height. In this grey-level relief region edges correspond to high watersheds and region compounds correspond to catchment basins, see fig. 2.6. The segmentation can be illustrated by filling this surface with water. Pixels belonging to the same catchment basin are homogeneous and represent a region [SHB99, p. 186].

2.4 Delaunay triangulation

The Delaunay triangulation is used to create a mesh of triangles from a set of points in a two- or three-dimensional space [GB98, p. 33].

2.4.1 Fundamental idea

The main-issue of the Delaunay triangulation is the constitution of triangles. Any triangle formed by its three corner points in a two-dimensional space has to have a circumcircle such that there are no other points located inside [GB98, p. 38], see fig. 2.7. Other points are only allowed at the perimeter.

In three dimensions instead of a circumcircle a circumscribed sphere has to be empty.

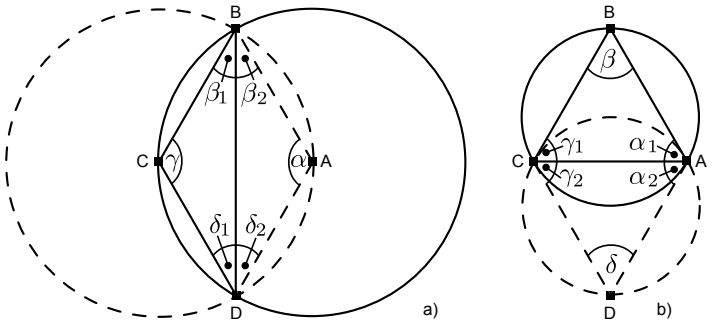

Figure 2.8: a) An arbitrary triangulation that does not fulfill the Delaunay condition of empty circumcircles. b) Flipping the common edge from BD to AC leads to a Delaunay triangulation.

2.4.2 Construction of a Delaunay triangulation

A well known method of the construction of a Delaunay triangulation is the diagonal swapping in two dimensions [GB98, p. 46].

First an arbitrary mesh of triangles is created. Afterwards it is checked if each triangle fulfills the Delaunay condition of the empty circumcircle. If the condition fails it always fails for a multiple of two triangles with a common edge, see fig. 2.8. Swapping the common edge diagonally creates two triangles that fulfill the Delaunay condition [GB98, p. 46].

From the edge swapping an important property concerning the angles can be derived: If the sum of the angles in opposite of the common edge of two triangles is less than or equal to 180° the triangles fulfill the Delaunay condition [GB98, p. 47], see fig. 2.8 for demonstration. In fig. 2.8 a) the sum of the angles $(\alpha + \gamma) > 180°$ while after flipping in fig. 2.8 b) the sum of the angles $(\beta + \delta) < 180°$.

This method can not be applied for a Delaunay triangulation in three dimensions. Although similar methods exist the construction of such a triangulation is not trivial [GB98, p. 48].

In this work the Delaunay triangulation is used to build the connected, undirected graph $G = (V, E)$ for the sparse optical flow segmentation.

Chapter 3

Graph-based segmentation

This chapter explains the implemented graph-based segmentation algorithms coded in C++. The segmentation of color images is described in sec. 3.1 and of optical flow images in sec. 3.2. Sec. 3.3 explains the algorithm if both images are used as input.

While the color segmentation needs only one color frame as input the segmentation of the optical flow images needs at least two (subsequent) grey-scale frames. Furthermore two different formulas for the calculation of the edge weights between neighboring nodes are introduced. In case of optical flow images the information of the direction of the motion vectors is optionally used.

Each section also provides an example output image in order to show how such a result looks like. A detailed comparison of the different algorithms for different image sequences is given in chapter 4.

3.1 Segmentation of a color image

This section describes the implemented segmentation algorithm for color images. First general definitions are introduced. Two different formulas for the calculation of the edge weight are highlighted and their pros and cons are briefly summarized. The cluster condition and its dependencies are discussed and finally the algorithm itself is described.

The C++ implementation of the segmentation algorithm was taken from [FH04] and is also explained in the following.

3.1.1 General definitions

Vertices $v_i \in V$ are defined that represent the pixels in the image which have to be segmented and edges $(v_i, v_j) \in E$ that connect vertices and neighboring pixels. Four edges per vertex[1] are sufficient to connect each vertex with all of its neighbors, see fig. 3.1. Vertices and edges together form the connected undirected graph $G = (V, E)$.

3.1.2 Edge weights

A non-negative weight $w((v_i, v_j))$ assigned to each edge measures the dissimilarity between the two vertices connected by that edge. There are different ways to determine the dissimilarity.

[1] In case of pixels located at the border of the image only edges to existing neighbors are defined.

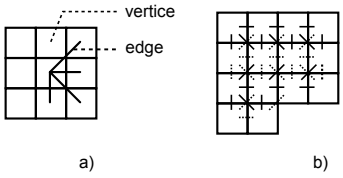

Figure 3.1: a) Definition of four edges for a single vertex. b) Four edges per vertex are sufficient to connect each vertex with all of its neighbors.

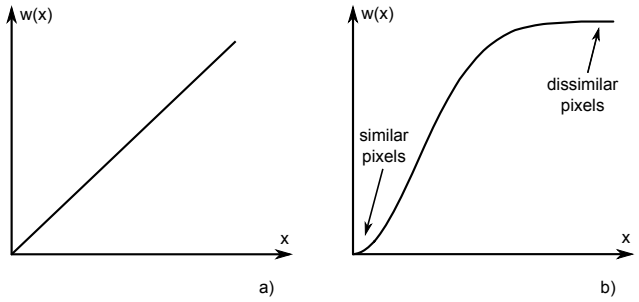

Figure 3.2: a) If the weight is calculated with the Euclidean distance the whole range is flat. b) The area of similar and dissimilar pixels are more dominant in case that the weight is calculated as exponential function.

[FH04] defines a weight formula such that the weight is low if the connected vertices are in the same component only. If not, the value becomes high.

Two different weight formulas have been implemented and tested. The first formula calculates the weight as Euclidean distance,

$$w_{euc} = \sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}, \tag{3.1}$$

where r_1, g_1 and b_1 are the color values of the first vertex and r_2, g_2 and b_2 are the respective values for the second vertex connected by that edge.

An alternative weight formula uses the exponential function that can be expressed as

$$w_{exp} = 1 - e^{-(\frac{w_{euc}}{\sigma_{rgb}})^2}, \tag{3.2}$$

where w_{euc} is the similarity of the vertices calculated with eq. 3.1 as Euclidean distance and σ_{rgb} defines the width of the Gauss-shaped curve. Tests yield a reasonable value of $\sigma_{rgb} = 44$, which is 10% of the maximum value[2] of x.

Compared to the Euclidean distance the exponential function has the advantage that the area of similar and dissimilar pixels are more dominant than the area in between, see fig. 3.2. This makes the splitting into components and therefore the segmentation more robust.

[2]Eq. 3.1 yields for two totally dissimilar vertices, $(r_1, g_1, b_1) = (0, 0, 0)$ and $(r_2, g_2, b_2) = (255, 255, 255)$ a maximum value of 441 and for two totally similar vertices $(r_1, g_1, b_1) = (r_2, g_2, b_2) = (255, 255, 255)$ a minimum value of 0.

3.1.3 Adaptive threshold and cluster condition

A *threshold function* $\tau(C)$ is used by the segmentation algorithm to decide wether pixels have to be joined to build a component or not. It is calculated as follows

$$\tau(C) = \frac{k}{|C|}. \tag{3.3}$$

k is a constant parameter specified by the user and $|C|$ is the size of the component. Note that $\tau(C)$ depends on the component size and therefore has to be re-calculated (adapted) with each joining.

In addition an *internal difference* $IntDif(C)$ for every component is defined as

$$IntDif(C) = \max_{e \in C} w(e), \tag{3.4}$$

which is the largest weight, e.g., the largest dissimilarity, within the component.

Using eq. 3.3 and 3.4 the *minimum internal difference* $MIntDif(C_i, C_j)$ between components is defined as

$$MIntDif(C_i, C_j) = \min[IntDif(C_i) + \tau(C_i), IntDif(C_j) + \tau(C_j)], \tag{3.5}$$

where C_i and C_j are the disjoint components. The segmentation algorithm uses the minimum internal difference in its cluster condition that has to be fulfilled to join two vertices.

The cluster condition[3] says

$$w((v_i, v_j)) \leq MIntDif(C_i, C_j) \ \wedge \ C_i \neq C_j \tag{3.6}$$

and in words:

- Weight $w((v_i, v_j))$ has to be smaller than the minimum internal difference $MIntDif(C_i, C_j)$ calculated with eq. 3.5 and
- vertices v_i and v_j have to be from different components C_i and C_j.

3.1.4 Algorithm

First the connected undirected graph $G = (V, E)$ has to be created from the input color frame and the resulting edges E have to be sorted in non-decreasing weight order.

Beginning with the smallest edge weight all edges are checked if they fulfill the cluster condition, eq. 3.6. If this is the case the two components are merged and the threshold function $\tau(C)$, the internal differences $IntDif(C)$ and the minimum internal difference $MIntDiff(C_i, C_j)$ are re-calculated. The algorithm continues with checking the next edge weight. If the cluster condition is not fulfilled the algorithm continues with the next edge.

After all edges have been checked a post-process for small[4] components starts. This process removes small regions by checking their size and merging them with one of their adjacent lager regions.

Fig. 2.3 shows an example output (segmentation parameters: edge weight = Euclidean distance, $k = 500$, $min = 50$). Pixels belonging to one component are given the same color for illustration.

[3] The cluster condition can be considered as the cost function introduced in sec. 2.3.2 to build a graph.

[4] The post-process removes components smaller than a given minimum number of vertices per component given by the user.

3.2 Segmentation using optical flow

This section describes the implemented segmentation algorithm for dense and sparse optical flow input images that can be computed from two grey-scale frames[5].

After making some general definitions[6] different edge weights, thresholds, a cluster condition and its dependencies are introduced and the algorithm itself is explained.

The magnitude and optionally the angle of the motion vector can be used. If only the magnitude of the motion vector is considered the definitions are similar to the definitions of sec. 3.1 for color images except that a pixel in the flow image represents velocity in x and y direction respectively and this denotes that the formulas have to be adjusted slightly.

If also the angle of the motion vector is applied the definitions have to be extended. An additional formula for a second edge weight and a second adaptive threshold are introduced. This asks for a extended cluster condition and modified dependencies that are described as well.

For the dense optical flow calculation an existing Win32 library and a sample program from [ZPB07] were taken. The computation of the sparse optical flow is implemented in C++ and can be found in my self-made class *ImageProcessingTools*. For the C++ implementation of the segmentation algorithm parts of the code from [FH04] were taken and extended.

3.2.1 Optical flow calculation

Dense optical flow calculation

For the calculation of the dense optical flow an existing Win32 library and a sample program from [ZPB07] were taken. Furthermore the sample program has been extended such that the computed dense optical flow is saved as binary *.flo* file, which is used as input for the segmentation algorithm.

Sparse optical flow calculation

The implemented C++ code computing the sparse optical flow image from two frames uses methods from Intel's Open Source Computer Vision (OpenCV) library [BK08]. First, this algorithm converts the color frames into grey-scale frames. After extracting the corner points from the first grey-scale frame those points are used to search for the appropriate corner points in the second frame. With the knowledge of the positions of the corner points in both frames the related motion vectors can be calculated which forms the sparse optical flow image.

It is important to consider, that sparse optical flow images are identical to dense optical flow images except that the motion vectors are only computed for certain (important) pixels in the image.

3.2.2 General definitions

The optical flow image is handled similar to the color image. The only difference is the meaning of their pixels. The pixels in the color image represent the r, g and b values, while the pixels in the dense optical flow image represent the motion vectors. In case of the sparse optical flow one has to consider that the motion vectors are not computed for all pixels in the image.

[5] If color images are applied as input they have to be converted into grey-scale images first.

[6] Wherever the definitions for the dense optical flow differ from the sparse optical flow images, a distinction is made.

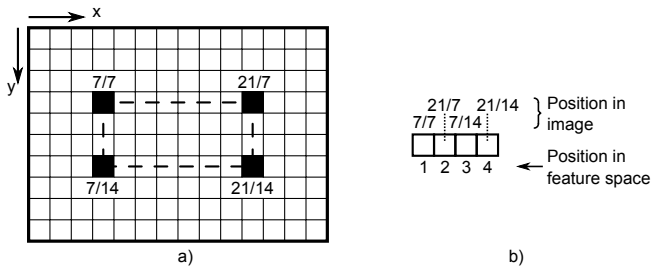

Figure 3.3: a) Synthetic image with non-neighboring corner points of a rectangle (dashed line). The positions are given as x/y coordinates of the real image. b) Pseudo-dense flow image with adjacent pixels after transformation into the *feature space*.

Vertices $v_i \in V$ for each pixel with computed motion vector can be defined. This implicates for sparse optical flow images that the total number of vertices is smaller than the number of pixels in the image.

Edges for dense optical flow

The edges $(v_i, v_j) \in E$ connecting vertices and their neighbors can be defined just as for color images. This means that four edges per vertex are sufficient to connect each vertex with all of its nine neighbors, see fig. 3.1.

Edges for sparse optical flow

For the sparse optical flow it is not assured that each vertex has a fixed number of neighbors. Beside this the location of the nearest neighbors is unknown. The neighbors can be determined using the Delaunay triangulation introduced in sec. 2.4 which assigns nearest neighbors to each vertex to constitute the edges $(v_i, v_j) \in E$.

Usually those neighbors are not nearby located but can be transformed into a so-called *feature space*. The *feature space* consists of adjacent pixels that can be mapped to the real pixels in the image, see fig. 3.3. The order of the mapping is not important. The major aim of this transformation is to obtain a pseudo-dense flow image in the *feature space*. This offers the possibility to use the segmentation algorithm as for real dense optical flow images in the end.

The C++ implementation of the determination of neighbor pixels can be found in my self-made class *ImageProcessingTools*.

3.2.3 Edge weights

Just as for the color image a weight $w((v_i, v_j))$ greater than or equal to zero can be assigned to each edge which measures the dissimilarity between the involved vertices. If the same weight definition is used (the weight between vertices in the same component is low and between vertices in different component is high) the same segmentation algorithm can be used as for the color image.

The weight formulas calculated as Euclidean distance (eq. 3.1) and exponential function (eq. 3.2) respectively using the motion vectors can be re-written as:

$$w_{euc} = \sqrt{(u_1 - u_2)^2 + (v_1 - v_2)^2} \text{ and} \tag{3.7}$$

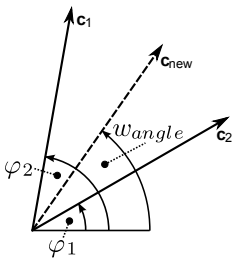

Figure 3.4: Second edge weight $w_{angle}((v_i, v_j))$ calculated as mean value of the connected motion vectors.

$$w_{exp} = 1 - e^{-(\frac{w_{euc}}{\sigma_{uv}})^2}, \tag{3.8}$$

where u_1 and v_1 are the changes of position in x and y of the first vertex respectively, u_2 and v_2 are the changes of position in x and y direction of the second vertex respectively connected by that edge, w_{euc} is the similarity of the vertices calculated with eq. 3.7 and σ_{uv} defines the width of the Gauss-shaped curve. Tests yield a reasonable value of $\sigma_{uv} = 3$, which is 1% of the maximum value[7] of x.

Eq. 3.7 and 3.8 only covers the magnitudes of the motion vector which additionally holds a direction. Therefore an additional edge weight $w_{angle}((v_i, v_j))$ using the directions can be defined and has been implemented as mean angle of the two motion vectors

$$w_{angle}((v_i, v_j)) = \frac{\varphi_1 + \varphi_2}{2} \tag{3.9}$$

where φ_1 and φ_2 are the angles of the two motion vectors connected by that edge (see fig. 3.4).

3.2.4 Adaptive thresholds and extended cluster condition

Due to the similarity between optical flow images and color images[8] the same definitions for the *threshold function*, the *internal difference* and the *minimum internal difference* as for the color image can be made.

Remember the *threshold function* $\tau(C)$ is calculated as

$$\tau(C) = \frac{k}{|C|}, \tag{3.3}$$

where k is a constant parameter specified by the user and $|C|$ is the size of the component.

The largest weight within the component called *internal difference* $IntDif(C)$ is defined as

$$IntDif(C) = \max_{e \in C} w(e). \tag{3.4}$$

[7]Both the Win32 library for the computation of the dense optical flow as well as the implemented code for calculating the sparse optical flow yield good results as long as the change in position is less than approximately 100 pixels. If the movement exceeds 100 pixels the computed results are wrong. Thus eq. 3.7 yields for two totally dissimilar vertices, $(u_1, v_1) = (0, 0)$ and $(u_2, v_2) = (100, 100)$ a maximum value of 282 and for two totally similar vertices $(u_1, v_1) = (u_2, v_2) = (100, 100)$ a minimum value of 0.

[8]Except that color pixels represent color values and dense optical flow pixels represent motion vectors.

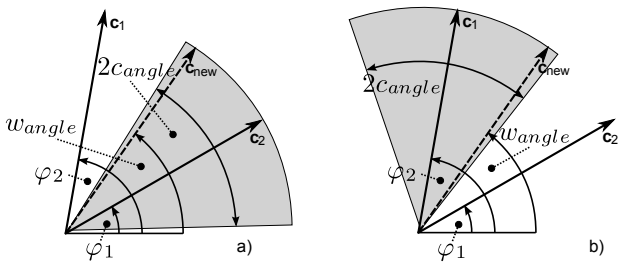

Figure 3.5: Illustrations of eq. 3.10. $w_{angle}((v_i, v_j))$ has to be located in both shaded areas. a) Tolerance around angle φ_1. b) Tolerance around angle φ_2.

The *minimum internal difference* $MIntDif(C_i, C_j)$ between components is expressed as

$$MIntDif(C_i, C_j) = \min[IntDif(C_i) + \tau(C_i), IntDif(C_j) + \tau(C_j)]. \tag{3.5}$$

Only if the angle of the motion vector is considered a second adaptive threshold for the angle of the components has to be defined and re-calculated after each merging.

The cluster condition for color images,

$$w((v_i, v_j)) \leq MIntDif(C_i, C_j) \ \wedge \ C_i \neq C_j, \tag{3.6}$$

has to be extended if the additional edge weight is used:

$$\begin{aligned}(\varphi_1 - c_{angle}) \leq w_{angle}((v_i, v_j)) < (\varphi_1 + c_{angle}) \wedge \\ (\varphi_2 - c_{angle}) \leq w_{angle}((v_i, v_j)) < (\varphi_2 + c_{angle}),\end{aligned} \tag{3.10}$$

where φ_1 and φ_2 are the angles of the two motion vectors and c_{angle} defines the tolerance around the angle φ_1 and φ_2 respectively where $w_{angle}((v_i, v_j))$ has to be located in.

Fig. 3.5 illustrates eq. 3.10 and together with 3.6 the cluster condition can be formulated in words as:

- Weight $w((v_i, v_j))$ has to be smaller than the minimum internal difference $MIntDif(C_i, C_j)$,
- vertices v_i and v_j have to be from different components C_i and C_j and
- the second edge weight $w_{angle}((v_i, v_j))$ has to be located within the areas defined by c_{angle}.

3.2.5 Algorithm

If the segmentation of the optical flow image ignores the direction information of the motion vectors the segmentation algorithm is totally equal to the segmentation algorithm of the color images (see section 3.1.4).

A better segmentation result can be obtained if the direction of the motion vectors is used. In this case the segmentation algorithm differs from the algorithm of color images. First the connected undirected graph $G = (V, E)$ has to be created in the same manner and the edges has to be sorted in non-decreasing weight order. The second edge weight $w_{angle}((v_i, v_j))$ that takes the direction of the motion vectors into account is just an extension for the cluster condition. For the sorting the first edge weight $w((v_i, v_j))$ dealing with the magnitude of the motion vector is of interest.

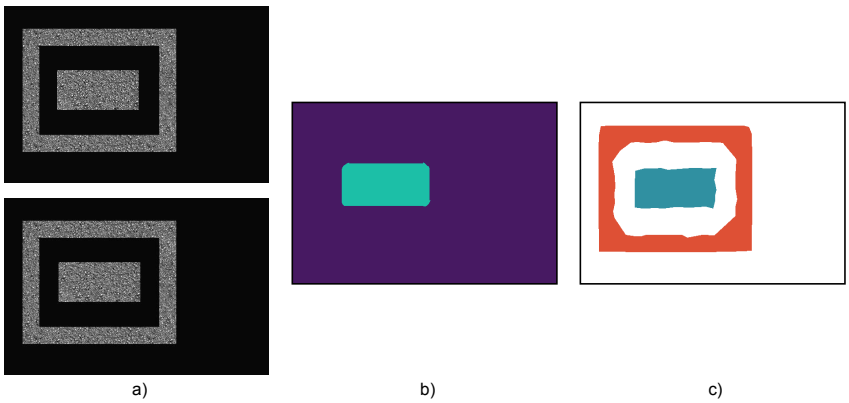

Figure 3.6: Segmentation output example. The inner box moves 5 pixels to the right. a) Input images at time t and $t+dt$ respectively. b) Segmentation output of the *dense* optical optical flow. The motion vectors are calculated for each pixel. Only the inner box moves. The output contains two objects, the moving box in cyan and the non-moving frame and background in violet. c) Segmentation output of the *sparse* optical optical flow. The motion vectors are calculated for each feature point. Feature points with the same motion vector are connected by lines which gives a set of triables that are filled. The output contains two components, the moving box in cyan and the non-moving frame in orange. Because the background (white) has no structure no feature points can be extracted.

Beginning with the smallest edge weight all edges are checked if they satisfy the extended cluster condition eq. 3.6 and 3.10. If this is the case the two components are merged and the threshold function $\tau(C)$, the internal differences $IntDif(C)$ and the minimum internal difference $MIntDiff(C_i, C_j)$ are re-calculated. In addition a new angle for the whole component calculated as mean value from all motion vectors that build this component is calculated. If the cluster condition fails the algorithm continues with the next edge.

After all edges have been checked a post-process for small[9] components starts. This process merges regions of small size to larger neighboring regions. In respect of the angle the smaller component gets the angle of the larger one.

In case of sparse optical flow input images the achieved output contains pixels located in the *feature space*. Therefore the real pixel coordinates in the image have to be recovered by a back transformation which gives a scatterplot.

Fig. 3.6 shows an artificial example output for a dense and a sparse optical flow input file (dense segmentation parameters: edge weight = Euclidean distance, angle of motion vector is ignored, $k = 200$, $min = 3$, sparse segmentation parameters: edge weight = Euclidean distance, angle of motion vector is ignored, $k = 500$, $min = 50$). For illustration pixels belonging to one component are given the same color. In the scatterplot of the segmented sparse optical flow image the non-neighboring pixels of one component are connected by lines which gives a set of triangles that are filled.

[9] If the component is smaller than a given minimum number of vertices per component given by the user.

3.3 Segmentation using color and optical flow information

This section is an extension to sec. 3.1 and 3.2 such that instead of a single color or optical flow image both are used as input images for the segmentation algorithm.

At least two color input frames are necessary for the computation of the optical flow. The optical flow image together with the first color image are used as input for further processing

Except the first edge weight, that has to consider the color information as well as the motion information in a single formula using exponential functions, this section makes the same definitions as for optical flow images. All equations are not just referred but also re-written for convenience.

3.3.1 Optical flow calculation

Dense optical flow calculation

As already introduced in sec. 3.2.1 for the computation of the dense optical flow the existing Win32 library and the modified sample program from [ZPB07] were used to create the binary *.flo* file used as one input for the segmentation algorithm.

Sparse optical flow calculation

For the computation of the sparse optical flow a C++ code was implemented that uses methods from the Open Source Computer Vision (OpenCV) library [BK08]. For an in-depth explanation of the algorithm please refer to sec. 3.2.1.

3.3.2 General Definitions

Vertices $v_i \in V$ can be defined in the same manner as for color or optical flow images. The only difference is that especially in the case of sparse optical flow images the count of vertices is less than the count of pixels in the image. Of course the meaning of the pixels of color and optical flow images is different and should be kept in mind but for the definition of the vertices only the count of potential pixels are of interest.

Edges for dense optical flow

Since the motion vectors of a dense optical flow image are calculated for each pixel in the image edges $(v_i, v_j) \in E$ can be defined in the same manner as for color images[10]. This means that four edges per vertex are sufficient to connect all pixels together, see fig. 3.1.

Edges for sparse optical flow

The computed sparse optical flow alone holds no information about the neighbors of a given vertex. They can be determined with a Delaunay triangulation, see fig. 3.2.2, that assigns nearest neighbors to the vertices. With this information edges $(v_i, v_j) \in E$ can be defined. Because the vetices are not adjacent a transformation of the scattered pixels into the *feature space* is recommended to get a pseudo-dense flow.

Each pixel in the color image holds color information while only certain pixels (corner points) in the sparse optical flow image hold a computed motion vector. For the combination of color

[10] All pixels in the color image hold a color information.

and motion information the values of the appropriate vertex positions specified by the sparse optical flow are taken.

3.3.3 Edge weight

A non-negative edge weight $w((v_i, v_j))$ can be assigned to each edge that mirrors the similarity between vertices connected by that edge. The weight formula, including the color and the motion information, is defined such that edges between vertices in different components are assigned a high weight value while vertices in the same component receive a low value.

The implemented formula that takes into account color and motion information can be expressed as

$$w = 1 - e^{-(\frac{w_{euc,rgb}}{\sigma_{rgb}})^2} e^{-(\frac{w_{euc,uv}}{\sigma_{uv}})^2}, \tag{3.11}$$

where $w_{euc,rgb}$ and $w_{euc,uv}$ are the similarities of the vertices of the color and the flow image respectively calculated as Euclidean distance,

$$w_{euc,rgb} = \sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2} \quad \text{and} \tag{3.1}$$

$$w_{euc,uv} = \sqrt{(u_1 - u_2)^2 + (v_1 - v_2)^2} \tag{3.7}$$

where r_1, g_1 and b_1 are the values of the red, green and blue fraction and u_1 and v_1 are the changes of position in x and y of the first vertex respectively, r_2, g_2, b_2, u_2 and v_2 are defined similar for the second vertex, σ_{rgb} and σ_{uv} define the width of the Gauss-shaped curve and were set to $\sigma_{rgb} = 44$ and $\sigma_{uv} = 3$ (see sec. 3.1.2 and 3.2.3 for details).

According to optical flow images a second edge weight $w_{angle}((v_i, v_j))$ dealing with the angle of the motion vector can be defined. As mentioned in sec. 3.2.3 it can be computed as mean angle,

$$w_{angle}((v_i, v_j)) = \frac{\varphi_1 + \varphi_2}{2} \tag{3.9}$$

where φ_1 and φ_2 are the angles of the two motion vectors connected by that edge, see fig. 3.4.

Adaptive threshold and cluster condition

The definitions of the threshold and the cluster condition are the same as for color or optical flow images (see sec. 3.1.3 and 3.2.4) and are highlighted in the following paragraph.

The *threshold function* $\tau(C)$ can be calculated as

$$\tau(C) = \frac{k}{|C|}, \tag{3.3}$$

where k is a constant parameter specified by the user and $|C|$ is the size of the component.

The *internal difference* $IntDif(C)$ is defined as the largest weight within the component,

$$IntDif(C) = \max_{e \in C} w(e). \tag{3.4}$$

The *minimum internal difference* $MIntDif(C_i, C_j)$ between disjoint components C_i and C_j is expressed as

$$MIntDif(C_i, C_j) = \min[IntDif(C_i) + \tau(C_i), IntDif(C_j) + \tau(C_j)]. \tag{3.5}$$

As long as the angle of the motion vector is used a second adaptive threshold is defined for each component and re-calculated after every joining.

The cluster condition considering only the color information and the magnitude of the motion vector can be formulated as for the segmentation of the color image

$$w((v_i, v_j)) \leq MIntDif(C_i, C_j) \ \wedge \ C_i \neq C_j \tag{3.6}$$

and is extended in the case that the angle of the motion vector is also considered to:

$$\begin{aligned}(\varphi_1 - c_{angle}) \leq w_{angle}((v_i, v_j)) < (\varphi_1 + c_{angle}) \wedge \\ (\varphi_2 - c_{angle}) \leq w_{angle}((v_i, v_j)) < (\varphi_2 + c_{angle}),\end{aligned} \tag{3.10}$$

where φ_1 and φ_2 are the angles of the two motion vectors and c_{angle} defines the tolerance around φ_1 and φ_2, see fig. 3.5.

3.3.4 Algorithm

If the resolution of the color image and the optical flow image differs the larger one is re-sized[11].

The segmentation algorithm for the combination of color and optical flow images is the same as for optical flow images and is explained briefly. In the following it is assumed that the angle of the motion vector is considered[12].

The connected undirected graph $G = (V, E)$ has to be created and the edges have to be sorted[13]. Beginning with the smallest edge weight the cluster condition 3.6 and its extension 3.10 are evaluated for each edge. If the equations are satisfied the two components are merged and the threshold function $\tau(C)$, the internal differences $IntDif(C)$ and the minimum internal difference $MIntDiff(C_i, C_j)$ are re-calculated. An angle calculated as mean value from all motion vectors that build this component is computed and assigned to that new component. If the cluster condition or its extension is not satisfied the algorithm continues with the next edge.

Finally a post-process is removing small regions by checking their size and merging them with one of their adjacent bigger regions. In respect of the angle the smaller component gets the angle of the bigger one.

The combination yields an output which is finer than the output of the segmentation of the dense optical flow and rougher than the segmentation output of the color image.

Fig. 3.7 shows an example for a color image together with an segmented dense optical flow and segmented sparse optical flow image respectively (dense segmentation parameters: angle of motion vector is ignored, $k = 3$, $min = 100$, $\sigma_{rgb} = 44$, $\sigma_{uv} = 3$, sparse segmentation parameters: angle of motion vector is considered, $k = 3$, $min = 5$, $\sigma_{rgb} = 44$, $\sigma_{uv} = 3$, $c_{angle} = 30°$). For visualization of the segmentation output a color value is assigned to the pixels whereas pixels of the same component hold the same color value. If the dense optical flow image was used as one input for the segmentation the result can be illustrated by directly plotting the pixels. If an sparse optical flow input image was used the edges have to be transformed into the *feature space* thus the real pixel coordinates have to be recovered by a back transformation. For better illustration pixels forming one component in this scatterplot are connected by lines which gives a set of triangles that are filled.

[11] Due to inaccuracies at calculation it can happen that the resolution of the computed optical flow image differs from the color input images (few pixels too many in width and height). The downsizing is implemented by removing columns (rows) at the end of the line (file). This can cause minor deflections which can be ignored.

[12] If the angle of the motion vector is ignored the extensions for the cluster condition and the calculation of the mean angle for the component are invalid.

[13] In non-decreasing weight order.

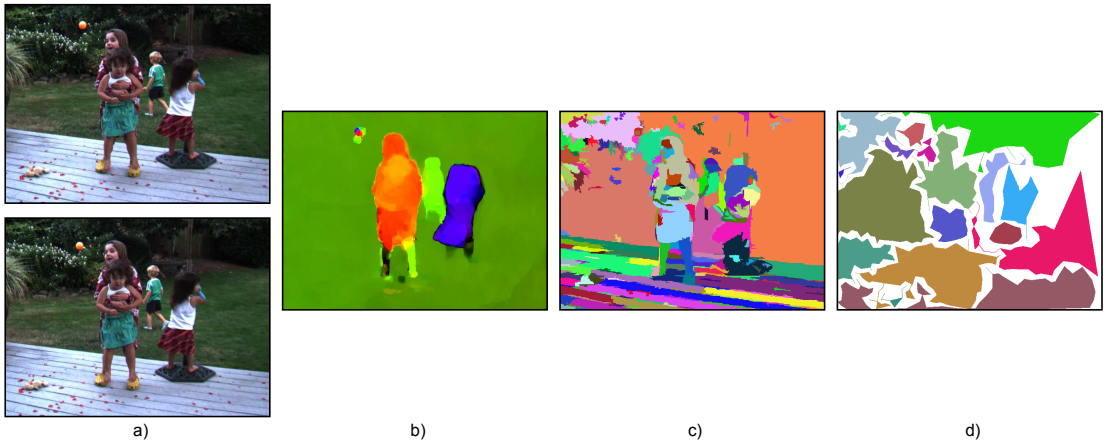

Figure 3.7: Segmentation output example. a) Color input images at time t and $t+dt$ respectively. b) Computed dense optical flow for images in a). c) Segmentation output with images in a) and b) as input. d) Segmentation output of the combination of color and sparse optical flow image. The conjunction of the color and the optical flow image refines the segmentation output of the optical flow image and roughs the segmentation output of the color image.

Chapter 4

Comparison and interpretation of the results

This chapter compares the results of the different segmentation algorithms using different input image sequences and discusses the results. First, sec. 4.1 analyzes the motions in the used datasets which helps to interpret the segmentation results. As extension the visualization of the dense optical flow images is introduced with the color circle in sec. 4.2. Sec. 4.3 and 4.4 discuss and qualitatively compare the segmentation results of the dense and sparse optical flow input images respectively using different edge weight formulas. The segmentation results compared among each other for same edge weight formulas are shown in sec. 4.5 and 4.6. Here the results are qualitatively as well as quantitatively compared[1]. Finally, sec. 4.7 briefly highlights the segmentation output of color and optical flow images during an excursus.

Concerning the segmentation parameters for each dataset the parameters were adjusted for the first frame to yield the best segmentation result. Using this parameters the rest of the frames were segmented. It can be assumed that this parameter sets can be used for any un-tested image sequence with similar content (e.g., same velocity of the objects, settings[2], frame-rate, image quality) because of their high similarity.

4.1 Motion analysis of color input datasets

For a serious comparison between the different segmentation algorithms standard datasets from a database[3] were used. Each dataset there contains eight frames. Additionally an image sequence from the lab with more than 30 frames was used. All of them are described in the following.

Dataset: Backyard

Motion characterization of the objects (order from left to right), see fig. 4.1:

- Ball: falls down to the ground; motion vector points downwards.

[1] The quantitative comparison was only drawn for the datasets *Backyard* and *Box06*.

[2] For example if the set of parameters was found for a road scenario it can be used for any other road scenario.

[3] Optical flow evaluation database of the Middlebury Computer Vision Page, `http://vision.middlebury.edu/flow/data/`.

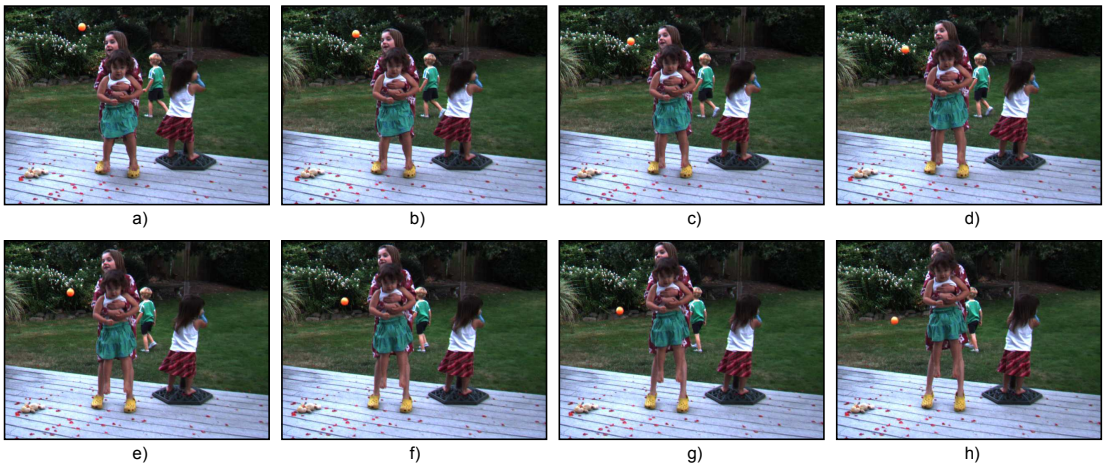

Figure 4.1: Dataset *Backyard* from the optical flow evaluation database. Camera and objects move. a) to h) Subsequent frames.

- Children in the left: jump into the air; a closer look shows that the motion vector of the bodies points upwards and of the legs to the left and the right respectively for the first frames.
- Child in the middle: walks from right to left; the motion vector points to the left.
- Child in the right: turns from left to right; the motion vector points to the right.
- Camera: moves slowly from left to right; the motion vector points to the left.

Because the camera moves in this sequence the total motion is a superposition of the movement of the objects itself and the camera.

Dataset: Dumptruck

Motion characterization of the objects (order from top to bottom), see fig. 4.2:

- Truck on the top: coasts from right to left; motion vector points to the left.
- Both cars in the middle: drive from left to right; motion vector points to the right.
- Car in the front: coasts into the image; motion vector points almost perpendicular to the drawing plane. A projection to the axis (assume that x and y span the drawing plane, z is perpendicular) yields a very small component in x- and y-direction.
- Camera: stands still.

Dataset: Box06

This dataset is an image sequence from the lab and consists of more frames than the dataset *Backyard* and *Dumptruck*. The image quality is worse than of the images from the database. For lack of space only eight images are shown in fig. 4.3. Its motion can be characterized as the following:

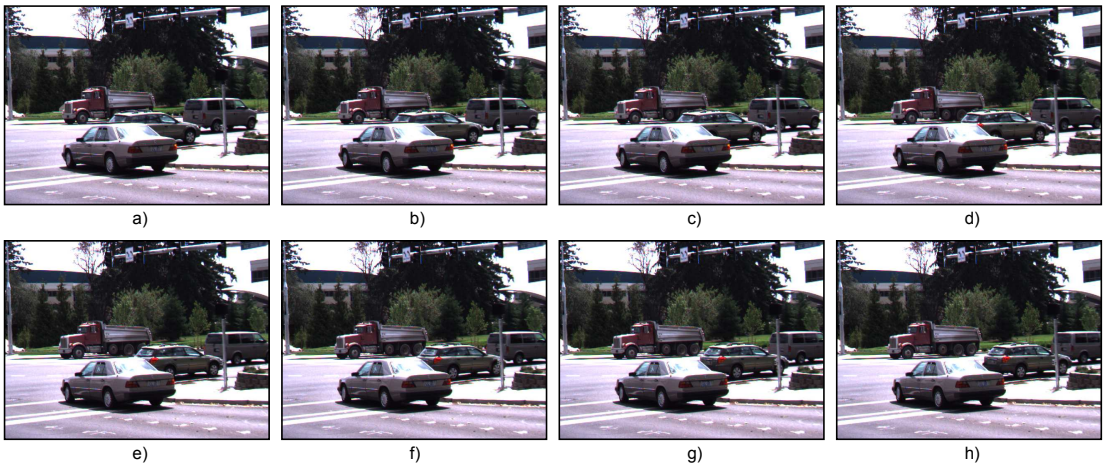

Figure 4.2: The dataset *Dumptruck* from the optical flow evaluation database. Camera stands still, objects move. a) to h) Subsequent frames.

- Box of the cookies: In the images displayed here the hand lifts the box. Its trajectory runs upwards following a circular path (that is more defined in the following frames).
- Camera: stands still.

4.2 Visualization of dense optical flow images

An introduction into the color representation of the computed dense optical flow image is given in the following.

A dense optical flow image is a binary image that contains motion vectors with certain magnitude and direction. A binary image can not be displayed in general. To make it visible a color mapping can be used. The color represents the direction and the intensity the magnitude of the motion vector. Fig. 4.4 shows the color circle that is used for definition. The less the saturation the smaller the magnitude. Black regions in the optical flow image correspond to motion vectors with evanescent magnitude. In those regions the magnitude is very small and not necessarily zero. This means that an angle for the motion vector can be computed that holds a random value in fact. This has to be handled by the segmentation algorithm.

Especially if the dense optical flow image contains regions where the optical flow can not be discovered correctly it can be helpful to have a closer look to the components of the flow. With the modified sample program from [ZPB07], as introduced in sec. 3.2, those images can be plotted as grey-scale images. For the u-component white and black represent a movement to the right and left, respectively. For the component v white and black represent movement downwards and upwards, respectively. Fig. 4.5 shows an example for the dense optical flow and its components computed for the first two frames of the dataset *Backyard*.

4.3 Dense optical flow images using different weight formulas

In this section the datasets described in sec. 4.1 were used to create segmentation results for different parameters. Concerning the edge weight that deals with the magnitude of the motion

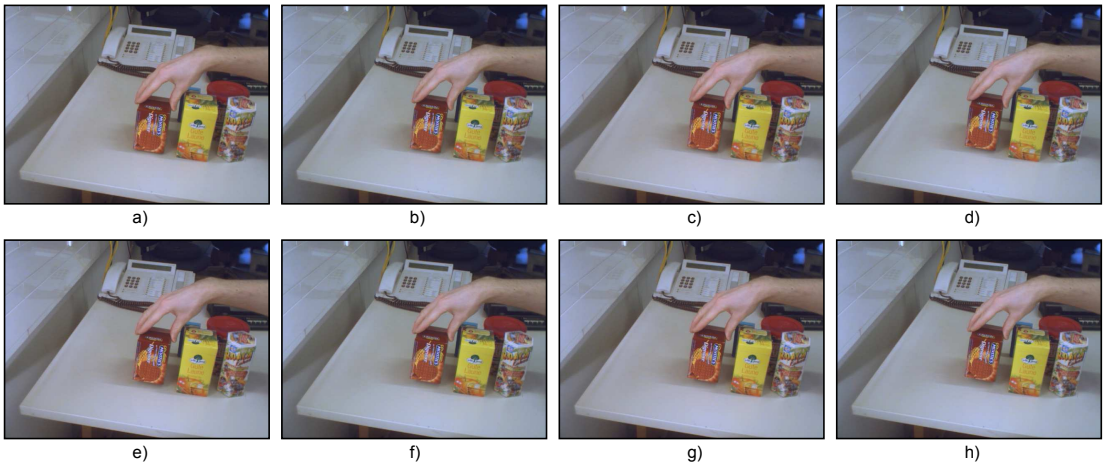

Figure 4.3: The dataset *Box06* from the lab. Camera stands still, objects move in linear and slightly circular motion. a) to h) Subsequent frames.

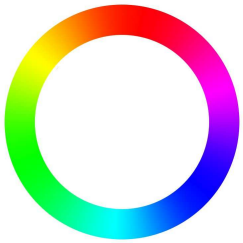

Figure 4.4: Color circle used to display the angle and magnitude of a motion vector. The color value represents the direction (=angle) and the intensity the magnitude.

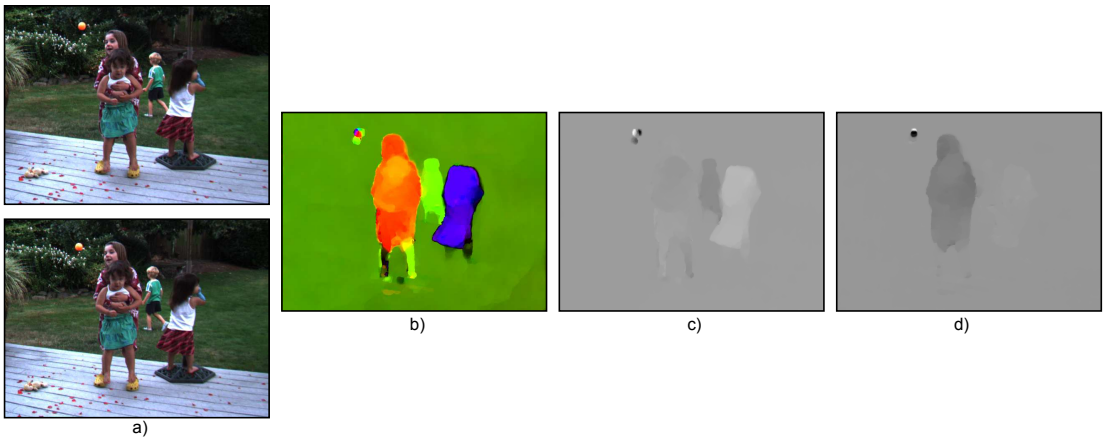

Figure 4.5: a) First two frames of the dataset *Backyard*. b) Computed dense optical flow for images in a). c) Component u of the dense optical flow. White and black represent a movement to the right and left, respectively. d) Component v of the dense optical flow. White and black represent movement downwards and upwards, respectively.

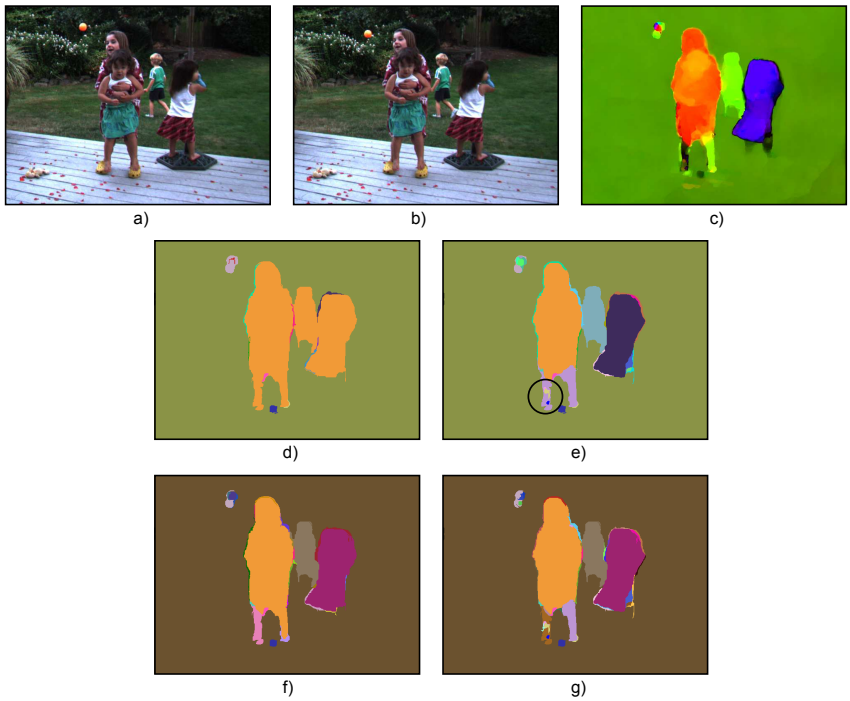

Figure 4.6: a) and b) First two frames of the dataset *Backyard*. c) Computed dense optical flow for images in a) and b). d) Segmented output (Euclidean distance, without angle). e) Segmented output (Euclidean distance, with angle). Marked deflection encircled. Error caused by the segmentation due to moderate flow computation encircled. f) Segmented output (Exponential function, without angle). g) Segmented output (Exponential function, with angle).

vector both the Euclidean distance and the exponential function were used, see sec. 3.2.3. If the angle of the motion vector was considered it was calculated as mean angle of the involved vectors.

The results of each dataset are qualitatively compared.

4.3.1 Dataset Backyard

Fig. 4.6 shows the first two color frames (a, b) that were used to calculate the dense optical flow[4] (c). Fig. 4.6 (d-g) show the resulting output images if the dense optical flow was used as input for segmentation.

Fig. 4.6 c) shows the computed dense optical flow image. Because in this dataset both the objects and the camera move the resulting motion vectors are a superposition of the objects movement and the movement of the camera[5]. The flow computation is inaccurate near object borders or for objects without texture, e.g. the ball in the upper left or the legs of the children in the left.

To generate the results the following sets of segmentation parameters were applied:

[4]Because the optical flow computation expects grey-scale images the input images from fig. 4.6 (a-b) were converted first.

[5]The theory concerning this superposition was introduced in sec. 2.1.2, *Optical flow in motion analysis*.

- Fig. 4.6 d): edge weight = Euclidean distance, angle of motion vector is ignored, $k = 110$, $min = 50$.
- Fig. 4.6 e): edge weight = Euclidean distance, angle of motion vector is considered, $k = 110$, $min = 50$, $c_{angle} = 20°$.
- Fig. 4.6 f): edge weight = Exponential function, angle of motion vector is ignored, $k = 3$, $min = 50$, $\sigma_{uv} = 3$.
- Fig. 4.6 g): edge weight = Exponential function, angle of motion vector is considered, $k = 3$, $min = 50$, $\sigma_{uv} = 3$, $c_{angle} = 20°$.

Comparison: Euclidean distance, with/without motion direction, fig. 4.6 d) and e)

In fig. 4.6 d) all (moving) objects from the input image are clustered together to form one component. This clustering is caused by the similar magnitude of the motion vectors of the children. By considering the angle, fig. 4.6 e), this component splits into parts as desired.

Note the area of the legs of the leftmost children in fig. 4.6 e): As can be seen in fig. 4.6 c) the direction of the motion is different for the right legs and the rest of the body. Thus the legs represent their own component. Especially the left legs should build a separate component too because the color and thus the direction is different from the right legs. This improper behavior is most likely caused by the segmentation algorithm in combination with the edge weight calculated as Euclidean distance. Using the exponential function for the weight calculation this effect disappears, see fig. 4.6 g).

Comparison: Exponential function, with/without motion direction, fig. 4.6 f) and g)

Compared to the weight computed as Euclidean distance, see fig. 4.6 d), here the results in fig. 4.6 f) already represents the objects from the input even if the angle of the motion vector is ignored. This means that the exponential function in combination with the used parameters is more sensitive. Tests showed that slightly altering the parameters k or σ_{uv} yield totally different segmentation output[6].

Because of the similar magnitude of the motion vectors of the leftmost children the body and the legs are clustered to one component in fig. 4.6 f) and split, because of the different directions in fig. 4.6 g). Additionally, the left legs of the leftmost children form their own component.

Comparison: Euclidean distance and exponential function, with motion direction, fig. 4.6 e) and g)

While the segmentation based on the edge weight calculated as Euclidean distance appears to have a lower performance in areas where the optical flow computation is ambiguous (see the area of the left legs of the leftmost children in fig. 4.6 c)) the weight formula using the exponential function gives better results.

[6] If the weight is calculated as Euclidean distance a small adjustment of those parameters does not have the same strong effects.

Summary

For this specific dataset the segmentation output using the exponential function for the edge weight calculation yields better results, especially in areas where the flow calculation is moderate or bad. However, the Euclidean distance has the advantage over the exponential function that the adjustment of the value k is easier to handle and thus it is preferable.

4.3.2 Dataset Dumptruck

Fig. 4.7 shows the used input frames (a-b), the computed dense optical flow (c) and the segmentation results (d-g).

Because in this image sequence the camera stands still the background of this image is colored black in the optical flow image in fig. 4.7 c). The magnitude of the computed motion vector is not exactly zero in this region but very small (e.g., < 0.003). Due to this non-vanishing magnitude an angle can be calculated for this motion vector witch holds a random value. The segmentation algorithm would cluster those non-zero motion vectors with angels to non-existing components. The workaround is the limitation of the magnitude. This is valid because it can be assumed that a computation of motion vectors with this accuracy is not possible. If it is smaller than a certain value both the magnitude and the angle will be set to zero. Within an existing component the magnitude of the motion vector can also vary slightly. If this magnitude is smaller than the threshold for the limitation the information of this pixel is removed. This causes a splitting of existing objects into several regions. See chapter 6.2 for improvements to handle this disadvantage.

For the segmentation following parameter sets were used:

- Fig. 4.7 d): edge weight = Euclidean distance, angle of motion vector is ignored, $k = 10$, $min = 450$.
- Fig. 4.7 e): edge weight = Euclidean distance, angle of motion vector is considered, magnitude limitation = 0.0075, $k = 10$, $min = 450$, $c_{angle} = 20°$.
- Fig. 4.7 f): edge weight = Exponential function, angle of motion vector is ignored, $k = 3$, $min = 50$, $\sigma_{uv} = 2$.
- Fig. 4.7 g): edge weight = Exponential function, angle of motion vector is considered, magnitude limitation = 0.003, $k = 3$, $min = 250$, $\sigma_{uv} = 2$, $c_{angle} = 20°$.

Because of the very small magnitude of the motion vectors of some of the cars the value k, used for the calculation of the threshold, had to be chosen very small. Since a small value of k causes many components the value min, used for the post-process to remove small components, had to be set accordingly high.

Comparison: Euclidean distance, with/without motion direction, fig. 4.7 d) and e)

In the segmentation output of fig. 4.7 d) and e) the form of the four major objects can be seen. They are represented as components split into several parts. Especially the car in the front and the truck in the back, that have very small motion vectors, are split inside their areas in fig. 4.7 e). This splitting is caused by the very restrictive limitation of the magnitude of the motion vector.

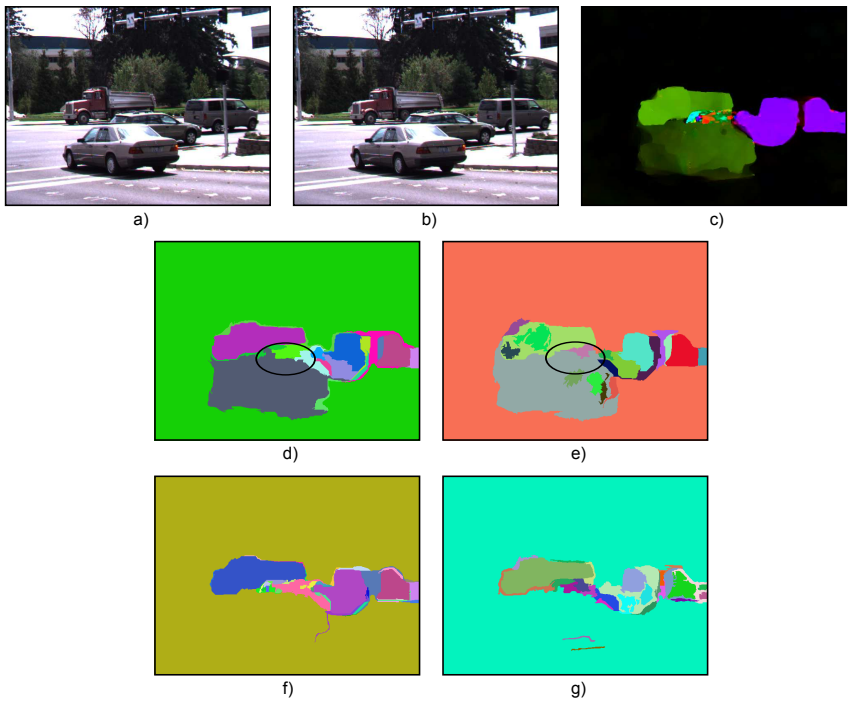

Figure 4.7: a) and b) First two frames of the dataset *Dumptruck*. c) Computed dense optical flow for images in a) and b). d) Segmented output (Euclidean distance, without angle). Marked deflection encircled. e) Segmented output (Euclidean distance, with angle). Marked deflection encircled. f) Segmented output (Exponential function, without angle). g) Segmented output (Exponential function, with angle).

Note the area of the windows of the left car in the middle (encircled in fig. 4.7 d) and e)). Those parts are clustered with the adjacent vehicles. This is because of the bad computation of the flow. Here the segmentation algorithm creates components that are very small and are removed by the post-process by merging them with the adjacent bigger areas.

Comparison: Exponential function, with/without motion direction, fig. 4.7 f) and g)

For the edge weight formula using the exponential function no parameter set could be found that makes the car in the front visible. A closer look to the movement of this car shows that the motion vector points almost perpendicular to the drawing plane. The components in x- and y-direction of the motion vector are very small[7]. The calculated weight is very small and is clustered with the background[8].

Also the splitting inside the components and frazzles caused by the limitation of the algorithm can be seen in the cars in the middle.

Comparison: Euclidean distance and exponential function, with motion direction, fig. 4.7 e) and g)

Here the usage of the exponential function has the disadvantage that the segmentation output does not contain all objects from the input image. The edge weight calculated using the Euclidean distance yields the better results for image sequences of this type (e.g., motion vectors that point into the drawing plane, very slow motions in general).

Summary

For objects with motion vectors that point almost perpendicular to the drawing plane the projection to the axis yields very small components for the x- and y-direction[9]. The edge weight calculated as Euclidean distance yields definitely better results.

4.3.3 Dataset Box06

Fig. 4.8 shows the first two color images (a, b) and the computed dense optical flow (c). The segmentation results for different edge weights and angle considerations are shown in (d-g).

Following sets of parameter were used to generate the segmentation results:

- Fig. 4.8 d): edge weight = Euclidean distance, angle of motion vector is ignored, $k = 60$, $min = 100$.
- Fig. 4.8 e): edge weight = Euclidean distance, angle of motion vector is considered, magnitude limitation = 0.07, $k = 60$, $min = 100$, $c_{angle} = 20°$.
- Fig. 4.8 f): edge weight = Exponential function, angle of motion vector is ignored, $k = 2$, $min = 100$, $\sigma_{uv} = 2$.

[7]The projection of this motion vector to the axis of a three-dimensional plane yields very small components for x and y and a large component for z (x and y span the drawing plane, z is perpendicular).

[8]Tests showed that the disappearance of the car in the front is not caused by the limitation of the magnitude.

[9]This assumes that x and y span the drawing plane.

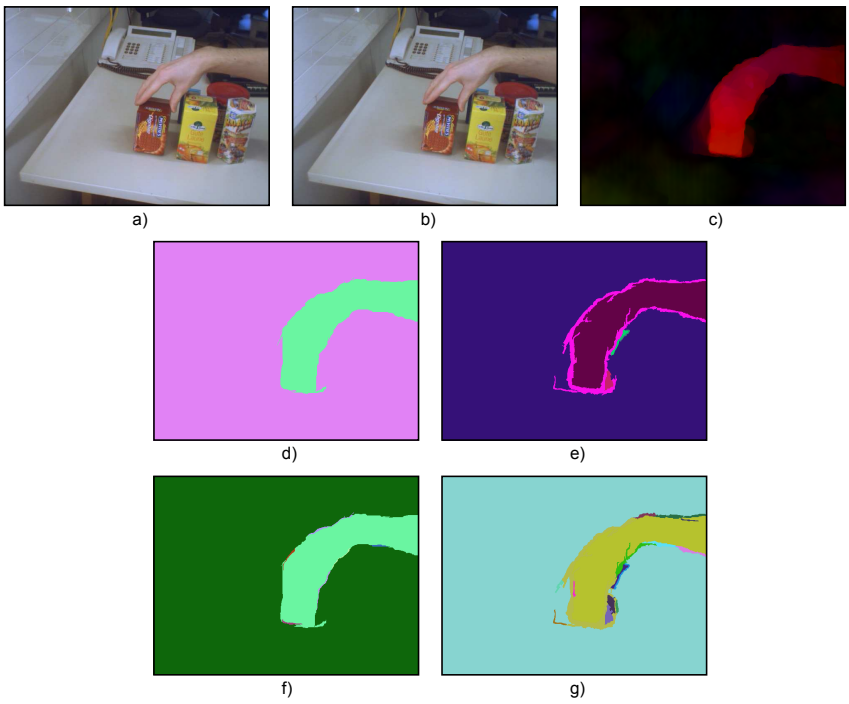

Figure 4.8: a) and b) First two frames of the image sequence *Box06* from the lab. c) Computed dense optical flow for images in a) and b). d) Segmented output (Euclidean distance, without angle). e) Segmented output (Euclidean distance, with angle). f) Segmented output (Exponential function, without angle). g) Segmented output (Exponential function, with angle).

- Fig. 4.8 g): edge weight = Exponential function, angle of motion vector is considered, magnitude limitation = 0.001, $k = 2$, $min = 100$, $\sigma_{uv} = 2$, $c_{angle} = 20°$.

Note that the camera does not move thus without the workaround, explained in sec. 4.3.2, the segmentation output would contain components in the background that do not exist. The magnitude of the motion vectors had to be limited. The magnitude and the angle of the vector are set to zero if its magnitude is smaller than the limitation value.

Comparison: Euclidean distance, with/without motion direction, fig. 4.8 d) and e)

The dense optical flow in fig. 4.8 c) shows a movement upwards. The form of the box together with the hand can be recognized quite good. Beside of the box the dense optical flow shows motion vectors computed for the shadows.

The frazzled object boundary in fig. 4.8 e) is caused by the limitation of the magnitude. Anyway, the form of the component is the same like in fig. 4.8 d). Here both results are the same independently from the angle. A difference can be expected if the scene contains at least two objects with similar velocity but in different directions. Ignoring the angles would cluster those components. They would not be split until the angle is considered. In general, for a more complex scene the segmentation using the direction information of the motion vector yields the better results.

Comparison: Exponential function, with/without motion direction, fig. 4.8 f) and g)

Calculating the edge weight with the exponential function makes the segmentation more sensitive. For example, the object border is more split in fig. 4.8 f) as well as in g). The form is similar to the one in the input image.

Comparison: Euclidean distance and exponential function, with motion direction, fig. 4.8 e) and g)

Fig. 4.8 g) contains more details, especially at the object border, as fig. 4.8 e), caused by the limitation of the magnitude. The forms of both segmentation results corresponds with the objects in the input image. Both edge weight formulas appear to be suitable for segmentation.

Summary

The components in the results correspond quite good with the objects in the input image. The qualitative comparison yields the result that both weight formulas create components that are frazzled at the object border. Whereas the weight computed as Euclidean distance in fig. 4.8 e) creates less deflections.

4.4 Sparse flow images using different weight formulas

In this section the segmentation output of sparse optical flow images for different edge weight formulas (Euclidean distance and exponential function, see sec. 3.2.3) are shown and qualitatively compared. If the angle was considered, it was calculated as mean angle of the motion vectors connected by that edge. The image sequences from sec. 4.1 were used as inputs.

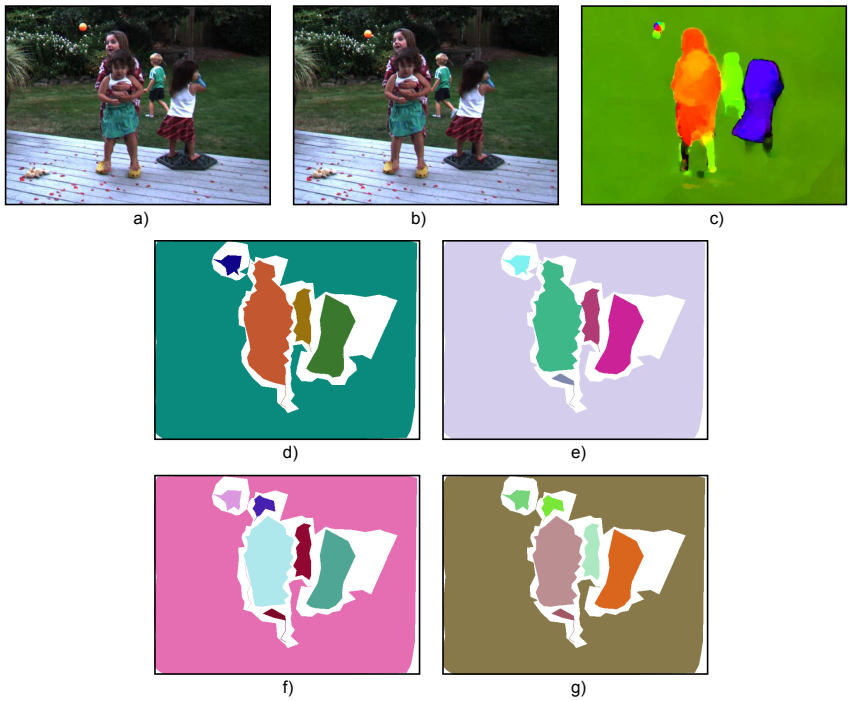

Figure 4.9: a) and b) First two frames of the dataset *Backyard*. c) Computed dense optical flow for images in a) and b). d) Segmented sparse optical flow output (Euclidean distance, without angle). e) Segmented sparse optical flow output (Euclidean distance, with angle). f) Segmented sparse optical flow output (Exponential function, without angle). g) Segmented sparse optical flow output (Exponential function, with angle).

4.4.1 Dataset Backyard

Fig. 4.9 shows the segmentation inputs and corresponding results. The first two color frames of the sequence are shown in fig. 4.9 (a-b). The computed dense optical flow for those input images are shown for the sake of completeness in fig. 4.9 c)[10]. Fig. 4.9 (d-g) show the results of the segmentation for different parameter sets. As input for the sparse optical flow segmentation acted the image sequence from fig. 4.1.

For the calculation of the sparse optical flow the corner points of an image were extracted. In case of objects that do not have a texture almost no corner points within the objects can be extracted. In this image sequence this is the case for the ball and the legs of the leftmost children. In most cases corner points for objects without texture can be extracted at the object boundary at the best. The assignment of those points to the object itself or its neighbor is difficult. If the corner points are assigned to the neighbor it can happen that the object itself disappears or parts of it are merged with the adjacent regions in the segmentation output.

To generate the results following segmentation parameters were used:

- Fig. 4.9 d): edge weight = Euclidean distance, angle of motion vector is ignored, $k = 90$, $min = 5$.

[10]The dense optical flow image highlights the components that should be included in the segmented sparse optical flow image.

- Fig. 4.9 e): edge weight = Euclidean distance, angle of motion vector is considered, $k = 90$, $min = 5$, $c_{angle} = 20°$.
- Fig. 4.9 f): edge weight = Exponential function, angle of motion vector is ignored, $k = 3$, $min = 5$, $\sigma_{uv} = 3$.
- Fig. 4.9 g): edge weight = Exponential function, angle of motion vector is considered, $k = 3$, $min = 5$, $\sigma_{uv} = 3$, $c_{angle} = 20°$.

Comparison: Euclidean distance, with/without motion direction, fig. 4.9 d) and e)

All objects from the input image are represented as components in the segmentation output.

The right leg of the leftmost child are clustered together with the body in fig. 4.9 d) because the magnitude of the motion vector is similar. If the direction is considered it forms its own component in fig. 4.9 e). This was already the case in fig. 4.6 d) and e).

The left legs of the leftmost children get lost anyway. This is caused by the corner points extracted only at the object border that are assigned to the background. The grid is spanned in front of the legs.

Comparison: Exponential function, with/without motion direction, fig. 4.9 f) and g)

Here the results differ slightly from the inputs. For example, the head of the leftmost child is separated as one component. The dense optical flow shows that it should be clustered with the body. Here the same effect as for the segmentation of the dense optical flow using the exponential function (sec. 4.3.1) appears.

Comparison: Euclidean distance and exponential function, with motion direction, fig. 4.9 e) and g)

Except the head of the leftmost child in fig. 4.9 g) that is split into its own component both results are equal.

Summary

Both segmentation results represent the objects from the input quite good. Even if the leftmost children are split into two components an additional post-process is needed anyway. The extracted corner points and thus the form of the segmentation results fit good to the form of the real objects. For image sequences where the objects and the camera move both edge weights yield good results.

4.4.2 Dataset Dumptruck

Fig. 4.10 (a-b) show the first two frame of the image sequence. In fig. 4.10 c) the computed dense optical flow of the images in fig. 4.10 (a-b) is shown. The images fig 4.10 (d-e) show the segmented output for the edge weight calculated as Euclidean distance with and without consideration of the motion direction. Fig. 4.10 (f-g) show the output if the exponential function is used. The inputs for the segmented sparse optical flow images are the color images in fig. 4.2.

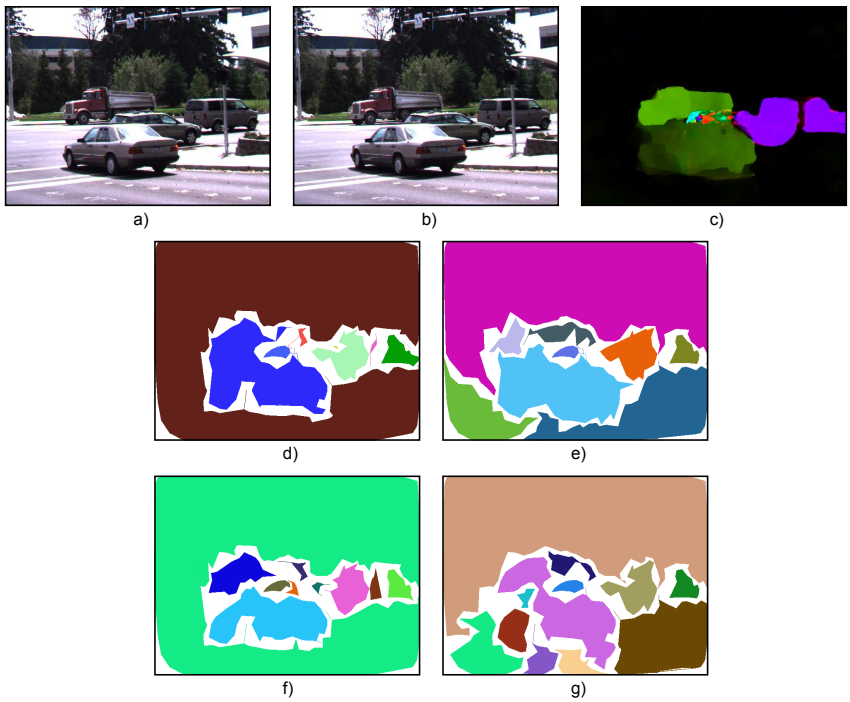

Figure 4.10: a) and b) First two frames of the dataset *Dumptruck*. c) Computed dense optical flow for images in a) and b). d) Segmented sparse optical flow output (Euclidean distance, without angle). e) Segmented sparse optical flow output (Euclidean distance, with angle). f) Segmented sparse optical flow output (Exponential function, without angle). g) Segmented sparse optical flow output (Exponential function, with angle).

As already mentioned in sec. 4.3.2 due to the non-moving camera the computed motion vectors for the background should be zero but are very small in fact. The limitation of the magnitude was also applied here. Those limitation causes splitting of the components, as it can be seen in fig. 4.10 e) and g), e.g., for the background or the truck in the back.

Following sets of parameters were used for the segmentation:

- Fig. 4.10 d): edge weight = Euclidean distance, angle of motion vector is ignored, $k = 30$, $min = 3$.
- Fig. 4.10 e): edge weight = Euclidean distance, angle of motion vector is considered, magnitude limitation = 0.2, $k = 30$, $min = 10$, $c_{angle} = 20°$.
- Fig. 4.10 f): edge weight = Exponential function, angle of motion vector is ignored, $k = 3$, $min = 5$, $\sigma_{uv} = 2$.
- Fig. 4.10 g): edge weight = Exponential function, angle of motion vector is considered, magnitude limitation = 0.005, $k = 3$, $min = 10$, $\sigma_{uv} = 2$, $c_{angle} = 20°$.

Comparison: Euclidean distance, with/without motion direction, fig. 4.10 d) and e)

The cars in the middle in fig. 4.10 d) as well as in e) are split into two components as it is the case in fig. 4.10 c). The splitting of the background and the truck in the back in fig. 4.10 e) is caused by the limitation of the magnitude (see sec. 4.3.2).

Comparison: Exponential function, with/without motion direction, fig. 4.10 f) and g)

In fig. 4.10 f) the truck in the back is split into several components while it should be represented as one component. This is caused by the segmentation parameters. With the used parameters the best segmentation output was achieved. The car in the front is represented as one component. As shown in fig. 4.7 f) or g) the segmented dense optical flow image could not detect the car in the front. The cars in the middle concerning their splitting as well as their forms look good. If the motion direction is considered (fig. 4.10 g)) the background is split due to the magnitude's limitation (see sec. 4.3.2). Additionally parts of the car in the front and the truck in the back are clustered together that should not be the case.

Comparison: Euclidean distance and exponential function, with motion direction, fig. 4.10 e) and g)

The segmentation using the Euclidean distance for edge weight calculation appears to has the better result (fig. 4.10 e)) than the segmentation using the exponential function. Objects from the input are split into several components but they can be merged by a post-process. The output in fig. 4.10 g) merges wrong components (parts of the truck in the back and of the car in the front). Even if the segmentation parameters are adjusted a better segmentation result can not be achieved.

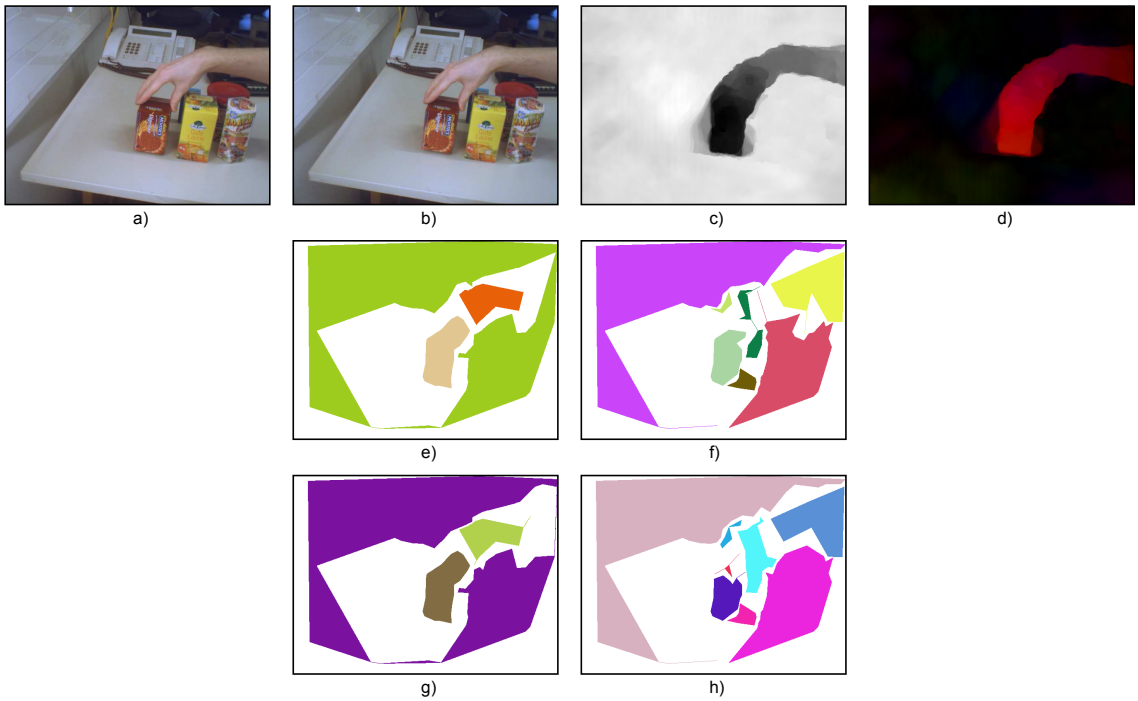

Figure 4.11: a) and b) First two frames of the image sequence *Box06* from the lab. c) Computed dense optical flow for images in a) and b). d) v-component of the dense optical flow in c). e) Segmented sparse optical flow output (Euclidean distance, without angle). f) Segmented sparse optical flow output (Euclidean distance, with angle). g) Segmented sparse optical flow output (Exponential function, without angle). h) Segmented sparse optical flow output (Exponential function, with angle).

Summary

For images sequences with moving and non-moving objects the segmentation of sparse optical flow images yields better results if the edge weight is calculated as Euclidean distance. The usage of the exponential function causes segmentation results with more details but that impairs the result.

4.4.3 Dataset Box06

Fig. 4.11 shows the first two color images (a, b) and the computed dense optical flow (c). The segmentation results for different edge weights and angle considerations are shown in (d-g).

Here the camera stands still which makes the usage of the workaround for limitation of the magnitude necessary. This limitation causes splitting of components that belong together. For an in-depth explanation of the workaround see sec. 4.3.2.

For the segmentation following sets of parameter were used:

- Fig. 4.11 d): edge weight = Euclidean distance, angle of motion vector is ignored, $k = 10$, $min = 5$.
- Fig. 4.11 e): edge weight = Euclidean distance, angle of motion vector is considered, magnitude limitation = 0.5, $k = 10$, $min = 5$, $c_{angle} = 20°$.

- Fig. 4.11 f): edge weight = Exponential function, angle of motion vector is ignored, $k = 3$, $min = 5$, $\sigma_{uv} = 2$.
- Fig. 4.11 g): edge weight = Exponential function, angle of motion vector is considered, magnitude limitation = 0.05, $k = 3$, $min = 5$, $\sigma_{uv} = 2$, $c_{angle} = 20°$.

Comparison: Euclidean distance, with/without motion direction, fig. 4.11 e) and f)

The non-moving objects (background, table and boxes in the right) are represented as one component in fig. 4.11 e) while they are split in fig. 4.11 f). Additionally in fig. 4.11 f) the shadow on the right side of the box as well as the fingers (as far as corner points are available) build their own component. This splitting is caused by the limitation of the magnitude (see sec. 4.3.2). The segmentation output in fig. 4.11 f) was the best one that could be achieved with the used parameter set.

In both results the hand is separated from the box. A closer lock to the u- and v-component, see especially fig. 4.11 d) for the v-component, of the dense optical flow shows that the components of the motion vector for the box and the hand are different. Therefore also the calculation of the angles for these objects is different and causes the splitting by consideration of the angle.

Because of the missing texture of the hand not enough corner points could be extracted to create a component with a form that looks more similar to the original one. This is a known problem but does not matter because in the overall vision framework a skin color detector is used that ignores skin colored regions and their unstable features[11].

Comparison: Exponential function, with/without motion direction, fig. 4.11 g) and h)

Fig. 4.11 h) shows the same high degree of splitting as 4.11 f). Here it is also caused by the limitation of the motion vector's magnitude . The segmentation output in fig. 4.11 g) looks quite good so it can be assumed that with an improved limitation-algorithm also by consideration of the angle the results will look better.

Comparison: Euclidean distance and exponential function, with motion direction, fig. 4.11 e) and g)

In this special case[12] the segmentation results of fig. 4.11 e) and g) are compared. Both results contain the the same moving objects as the input. The very small different in the motion vectors of the box and the hand can be used to split them into two components.

Summary

Both formulas for calculating the edge weights appear to yield very good segmentation results. To find out which formula is the better one a segmentation over several frames has to be made.

[11] For example, if a robot grasps at an object it knows its hand and can remove it from the image that has to be segmented.

[12] One object (hand and box) move in one direction. The consideration of the angle does not change the segmentation result.

4.5 Dense and sparse flow images using Euclidean distance

In this section the datasets from sec. 4.1 were used. While the parameters for the segmentation of the first dense optical flow and sparse optical flow images respectively were chosen such that the resulting segmentation output was best, those parameter sets were used for the segmentation of the remaining flow images. In the following the segmentation results for dense and sparse optical flow images for both edge weight formulas considering the direction of the motion vector are compared.

For details of the used parameter sets for the segmentation of dense optical flow files please refer to the appropriate dataset in sec. 4.3. For the segmentation of sparse optical flow images the parameters from the appropriate dataset in 4.4 were used.

4.5.1 Dataset Backyard

Fig. 4.12 shows the results of the segmentation of dense optical flow and sparse optical flow images. The images in fig. 4.12 a) are the inputs for the dense optical flow segmentation while the inputs for the sparse optical flow segmentation are the color images from fig. 4.1.

Qualitative comparison

Here the results of the segmentation of dense optical flow and sparse optical flow images were qualitatively compared with the computed dense optical flow input image and described in the following table. The first column contains the items that were checked. The second and third column contain the comparison results between the output images in fig. 4.12 b) and c) respectively and the input image in fig. 4.12 a).

Item of comparison	Segm. dense OF	Segm. sparse OF
All components exist	6/7	6/7
1 component lost	1/7	1/7
2 components merged into 1	1/7	1/7
1 components split in several parts	3/7	4/7

Especially the comparison of the output from the segmentation of the sparse optical flow with the computed dense optical flow is allowed because the latter acts as a guideline. As already mentioned, in the sparse optical flow image the motion vectors are computed for corner points. For regions with missing texture (e.g., the legs in this example) the extraction of corner points is difficult and a known problem. For this region corner points are extracted basically for the border and not for the object body thus the object itself can get lost. However, to draw a reasonable comparison those regions in the image are omitted. In particular this is the ball and the legs of the leftmost children.

The segmentation results are very similar. While in fig. 4.12 b) the component in frame three gets lost it is contained in the corresponding sparse segmentation output, fig. 4.12 c). On the other hand the component that get lost in frame seven in fig. 4.12 c) exists in fig. 4.12 b). Also the behavior concerning splitting is similar.

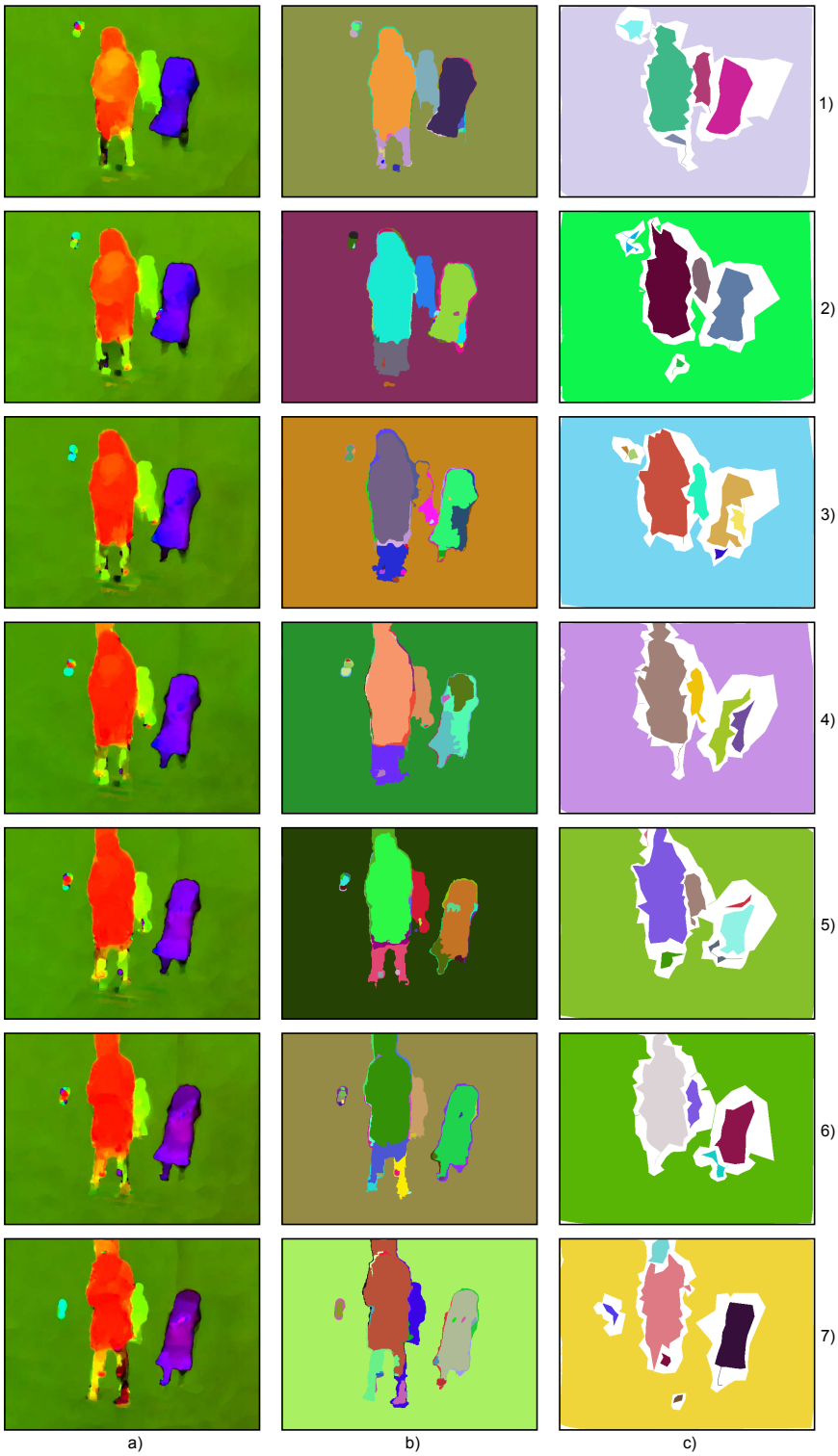

Figure 4.12: Segmentation results of dense and sparse optical flow images. Edge weight was calculated as Euclidean distance and the angle was considered. a) Computed dense optical flow images from the dataset *Backyard* for the input images in fig. 4.1. b) Segmented dense optical flow after using images from a) as input. c) Segmented sparse optical flow after converting images from fig. 4.1 into grey-scale images and computing the sparse optical flow used as input for the segmentation.

Quantitative comparison

For this comparison frame number two in fig. 4.12 was analyzed. The legs of the left children and the ball were not considered. The three objects from the color image and the corresponding components from the dense and sparse optical flow image were cut out. A degree of matching between each object and its corresponding component was calculated using following formula:

$$Match = \frac{I_{col}(C_i) \wedge I_{flo}(C_i)}{I_{col}(C_i) \vee I_{flo}(C_i)} \cdot 100\%, \tag{4.1}$$

where $I_{col}(C_i)$ is a binary image that contains one object from the color image, $I_{flo}(C_i)$ is a binary image that contains the corresponding component from the segmented optical flow image. In the binary images the pixels forming the object or component respectively have the value 1 (other pixels have zero value). The logic operation $\wedge$ (conjunction) is applied on every pixel in the images and the result value is 1 if both pixels have the value 1, otherwise zero. The logic operation $\vee$ (disjunction) is also applied on every pixel and the result value is 1 if at least one pixel has the value 1, otherwise zero. The number of pixels with the value 1 within the images are counted and the ratio is computed.

To determine the information about the segmentation performance for the whole frame the mean value of the partial results is calculated.

The following table summarizes the results. The first column shows the object that was selected. The second and third column show the degree of matching for the dense and sparse optical flow image, respectively.

Object of comparison	Match (dense OF)	Match (sparse OF)
children, left	77.6%	74.1%
child, middle	61.4%	58.4%
child, right	72.9%	75.6%
image	70.6%	69.4%

Summary

The segmentation performance of dense and sparse optical flow images yield equally satisfactory results. However, the latter implementation is preferable because of its lower demand for computational power. The ideal value of 100% is never reached because only parts of the compared object move.

4.5.2 Dataset Dumptruck

Fig. 4.13 a) shows the dense optical flow images. The corresponding segmentation results are shown in fig. 4.13 b). The results of the segmentation of sparse optical flow images can be seen in fig. 4.13 c).

Qualitative comparison

For the comparison the splitting caused by the limitation of the magnitude was ignored. It can be distinguished from the splitting caused by a small value of k because the borders are frazzled.

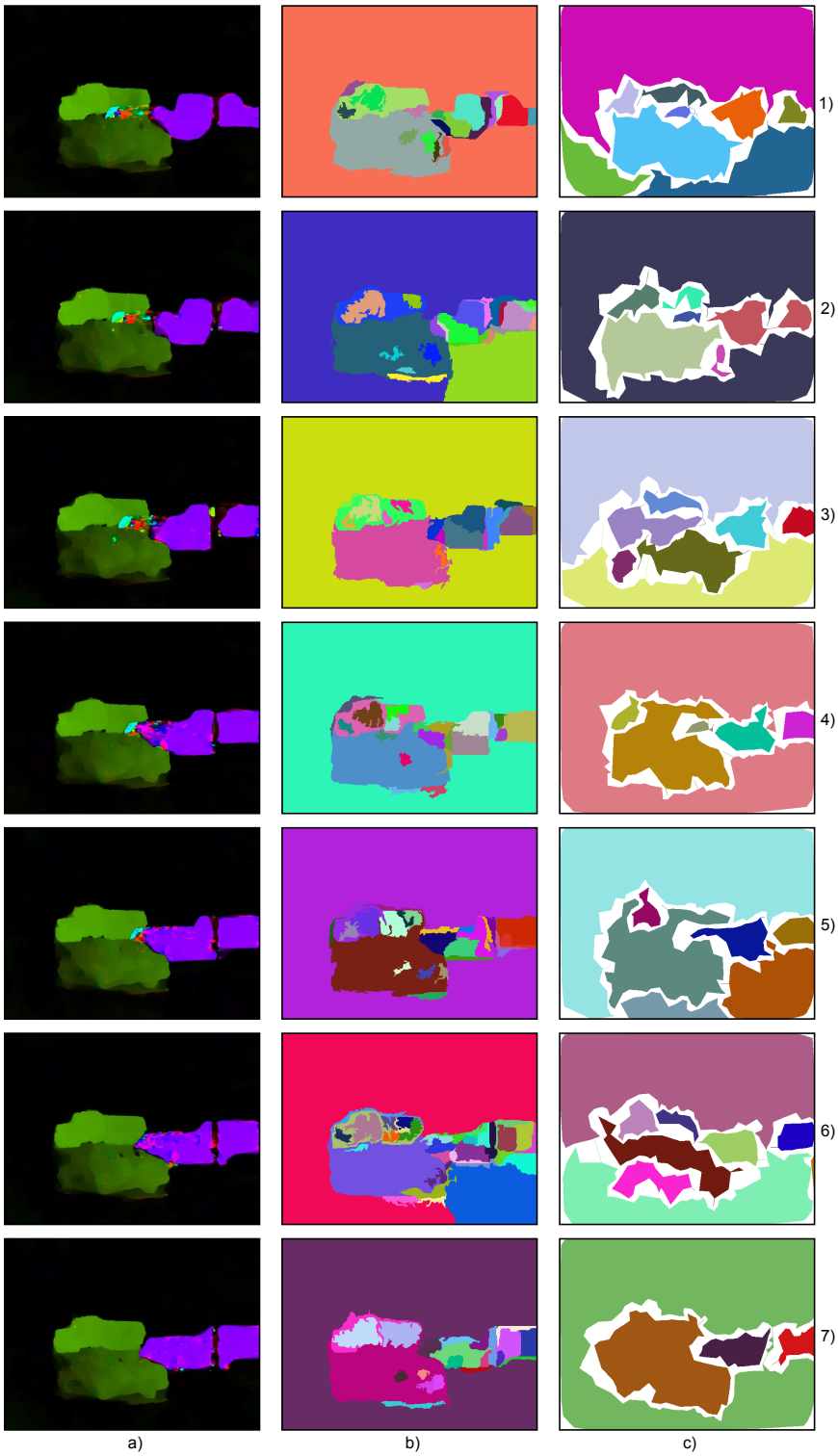

Figure 4.13: Segmentation results of dense and sparse optical flow images. Edge weight was calculated as Euclidean distance and the angle was considered. a) Computed dense optical flow images from the dataset *Dumptruck* for the input images in fig. 4.2. b) Segmented dense optical flow after using images from a) as input. c) Segmented sparse optical flow after converting images from fig. 4.2 into grey-scale images and computing the sparse optical flow used as input for the segmentation.

The results of a qualitative comparison of the segmented dense optical flow images and segmented sparse optical flow images respectively with the dense optical flow image are given in the next table. The first column contains the items that were checked. The second and third column contain the comparison results between the output images in fig. 4.13 b) and c) respectively and the input image in fig. 4.13 a).

Item of comparison	Segm. dense OF	Segm. sparse OF
All components exist	7/7	7/7
1 component lost	0/7	0/7
2 components merged into 1	0/7	3/7
1 components split in several parts	7/7	6/7

Unfortunately almost all components, that should represent one object from the input image, are split into several parts (note, that the splitting caused by the algorithm is ignored). This is caused by the very small value for k, which is necessary to make the car in the front visible. In fig. 4.13 b) the cars in the middle are split into several parts though the flow computation looks good. The truck and the car in the front form one component each. In fig. 4.13 c) it is the other way around. The cars in the middle form their own component as in the dense optical flow image while the other vehicles are broken apart.

Summary

This image sequence is difficult to analyse because of the slow motions of the objects. The segmentation of both dense and sparse optical flow images yield similar results. From the object detection point of view the sparse optical flow image is preferable because its output is less detailed than the segmented dense optical flow.

4.5.3 Dataset Box06

The dense optical flow images used as one input for the segmentation are shown in fig. 4.14 a). The corresponding results can be seen in fig. 4.14 b). From the input images in fig. 4.3 the sparse optical flow was computed and segmented. Those results are shown in fig. 4.14 c).

Different to the datasets *Backyard* and *Dumptruck* here the results for the segmentation ignore the direction of the motion vector. If the angle is considered the workaround for the limitation of the smallest magnitude is necessary. This limitation causes a splitting of components that makes the evaluation of the segmentation results quite difficult. In this image sequence the moving objects move into the same direction. Additionally the segmentation of the dense optical flow and sparse optical flow used the same parameter set even if the angle was considered or not. So the results are the same and this justifies the approach to compare the results with ignored direction information of the motion vector.

Qualitative comparison

For the qualitative comparison of the segmentation results in fig. 4.14 b) the dense optical flow images were used. The segmentation results of fig. 4.14 c) were compared with the color images in fig. 4.3.

This image sequence contains hundreds of color and optical flow images. The optical flow computed for eight subsequent frames of them are shown in fig. 4.14 a).

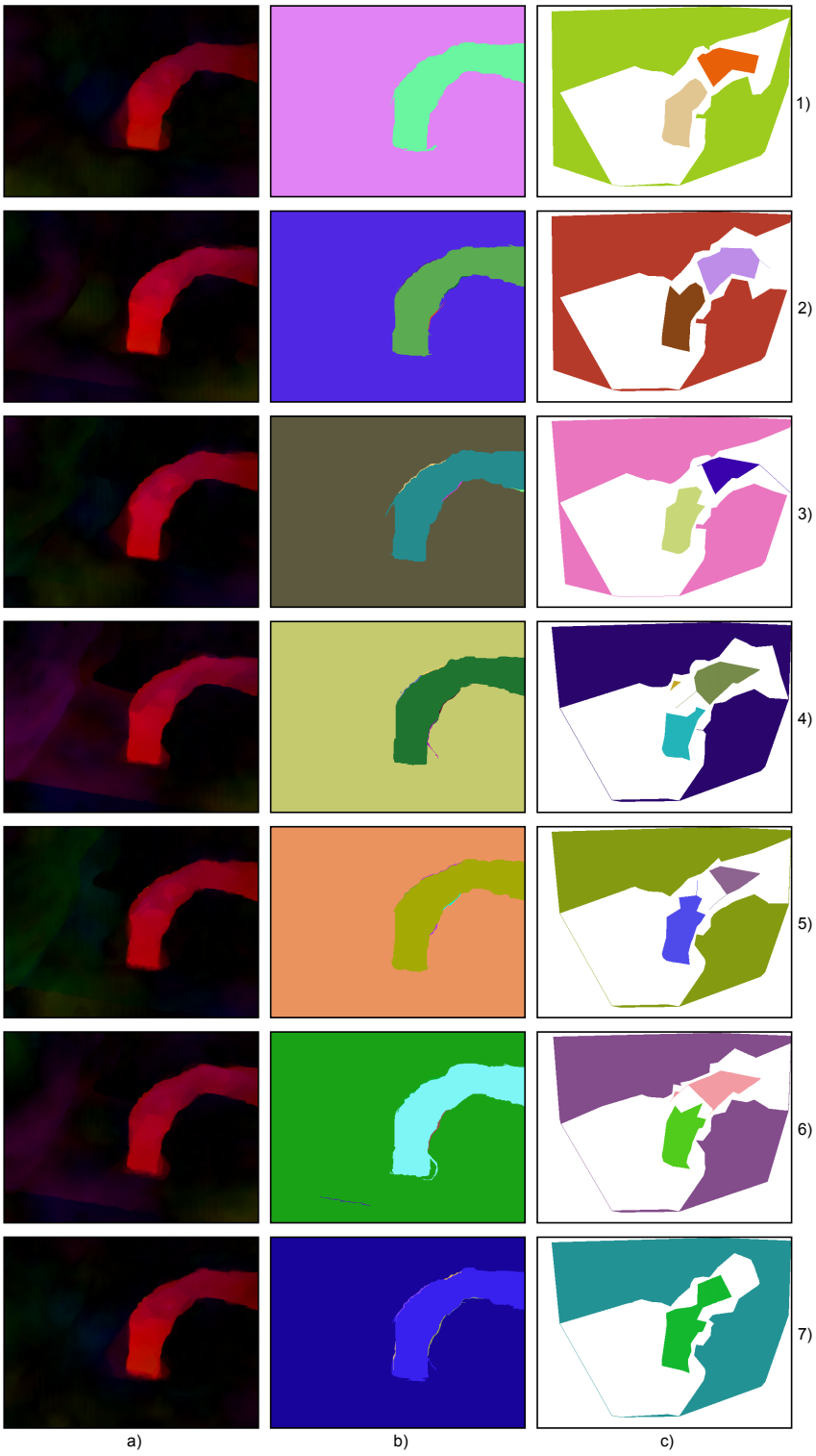

Figure 4.14: Segmentation results of dense and sparse optical flow images. Edge weight was calculated as Euclidean distance and the angle was ignored. a) Computed dense optical flow images from the dataset *Box06* for the input images in fig. 4.3. b) Segmented dense optical flow after using images from a) as input. c) Segmented sparse optical flow after converting images from fig. 4.3 into grey-scale images and computing the sparse optical flow used as input for the segmentation.

Following table holds the results of the comparison. The first column contains the items that were checked. The second column contains the comparison results between the output images in fig. 4.14 b) and the input image in fig. 4.14 a). Column three contains the results between images in fig. 4.14 c) and input images in fig. 4.3.

Item of comparison	Segm. dense OF	Segm. sparse OF
All components exist	7/7	7/7
1 component lost	0/7	0/7
2 components merged into 1	7/7	0/7
1 components split in several parts	0/7	3/7

The form of both segmentation results look good. The segmented sparse optical flow has the advantage that each moving object from the input image is represented as a single component in the output.

Quantitative comparison

For this analysis frame number two in fig. 4.14 was observed. The hand was not considered for this comparison. The box from the color image and the corresponding component from the dense and sparse optical flow image were cut out. The degree of matching was calculated using eq. 4.1.

The following table shows the results. The first column indicates the object that was selected. The second and third column show the degree of matching for the dense and sparse optical flow image, respectively.

Object of comparison	Match (dense OF)	Match (sparse OF)
box=image	76.6%	88.9%

Summary

While the computed dense optical flow shows that the box and the surrounding shadow are defined as one object, the corner points extracted for the sparse optical flow computation define only the box. Thus the segmentation result of the sparse optical flow image yields the clearly better result. Since the object borders are only approximated the ideal value of 100% is never reached.

4.6 Dense and sparse flow images using exponential function

4.6.1 Dataset Backyard

The results of the segmentation of dense optical flow and sparse optical flow images can be seen in fig. 4.15 b) and c) respectively. Fig. 4.15 a) shows the computed dense flow images that were the inputs for the segmentation of the dense optical flow. The color images in fig. 4.1 were the inputs for the sparse flow segmentation.

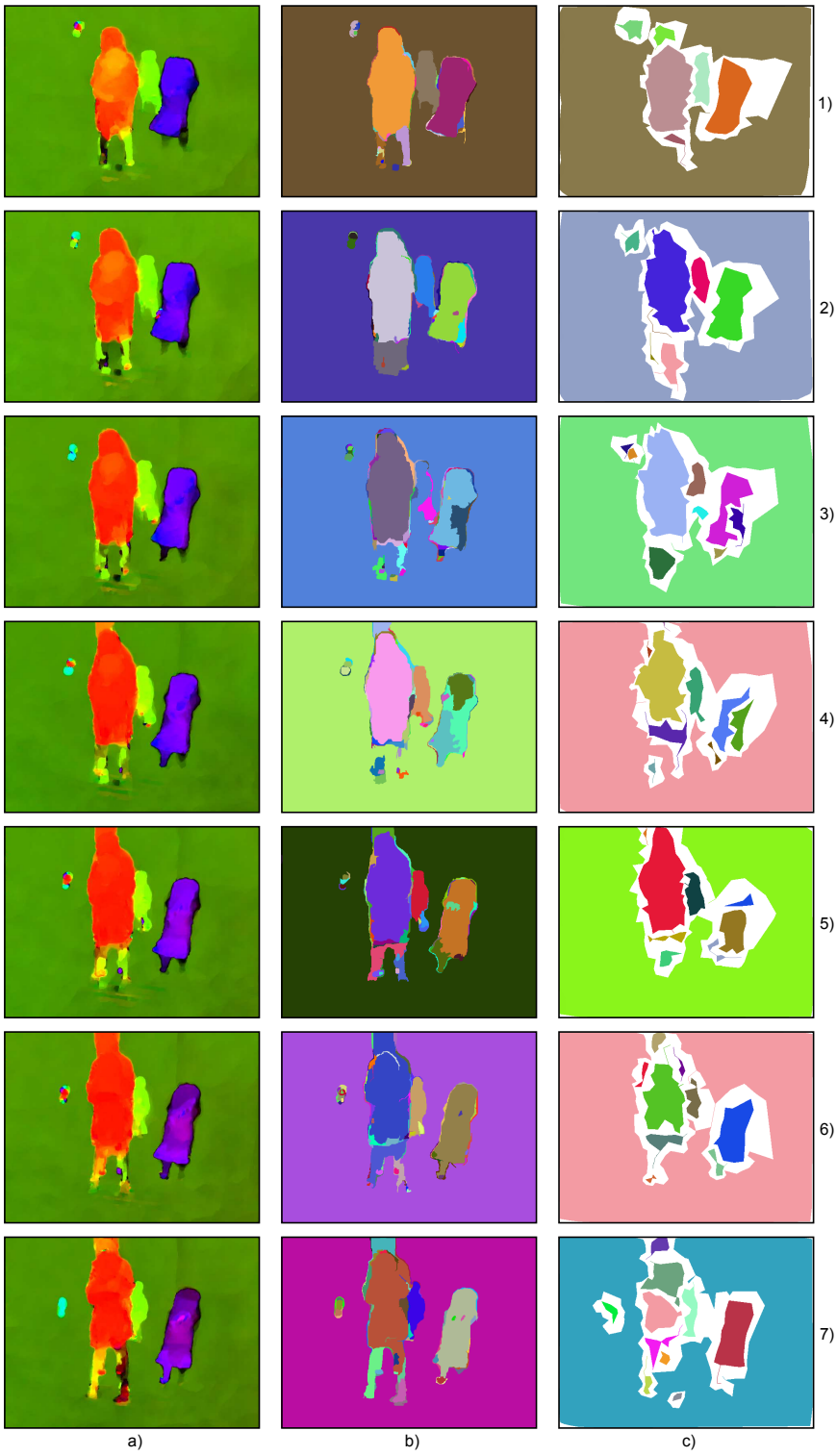

Figure 4.15: Segmentation results of dense and sparse optical flow images. Edge weight was calculated as exponential function and the angle was considered. a) Computed dense optical flow images from the dataset *Backyard* for the input images in fig. 4.1. b) Segmented dense optical flow after using images from a) as input. c) Segmented sparse optical flow after converting images from fig. 4.1 into grey-scale images and computing the sparse optical flow used as input for the segmentation.

Qualitative comparison

The following table shows the results of the qualitative comparison between the segmented dense optical flow and segmented sparse optical flow images respectively and the computed dense optical flow input images. The first column of the table contains the items that were checked while the second and third column contain the comparison results between the output images in fig. 4.15 b) and c) respectively and the input image in fig. 4.15 a).

Item of comparison	Segm. dense OF	Segm. sparse OF
All components exist	6/7	7/7
1 component lost	1/7	0/7
2 components merged into 1	0/7	1/7
1 components split in several parts	3/7	4/7

As already mentioned in sec. 4.5.1 because a sparse optical flow is computed for corner points in an image especially regions in the image without corner points have to be omitted for this comparison. In particular this is the ball and the legs of the leftmost children.

The segmentation results are very similar. One difference is the missing component in frame three of of fig. 4.15 b) which exists in c). Also the splitting behavior is similar.

Quantitative comparison

In this inspection the legs of the left children and the ball were not considered. The three objects from the color image and the corresponding components from the dense and sparse optical flow image were cut out from frame number two in fig. 4.15. For the calculation of the degree of matching eq. 4.1 was used. The overall segmentation performance for the whole frame was calculated as mean value of the partial results.

The following table summarizes the results. The first column shows the object that was selected. The second and third column show the degree of matching for the dense and sparse optical flow image respectively.

Object of comparison	Match (dense OF)	Match (sparse OF)
children, left	79.2%	69.8%
child, middle	61.8%	58.4%
child, right	76.6%	75.6%
image	72.5%	67.9%

Summary

Here, the segmentation of the dense optical flow is slightly better than the segmentation of the sparse optical flow. Anyway, the ideal value of 100% is not reached.

4.6.2 Dataset Dumptruck

The segmentation inputs and their results can be seen in fig. 4.16. The corresponding segmentation results of the dense optical flow images (fig. 4.16 a)) can be seen in fig. 4.16 b). The segmentation results in fig. 4.16 c) used the images from fig. 4.2 as input.

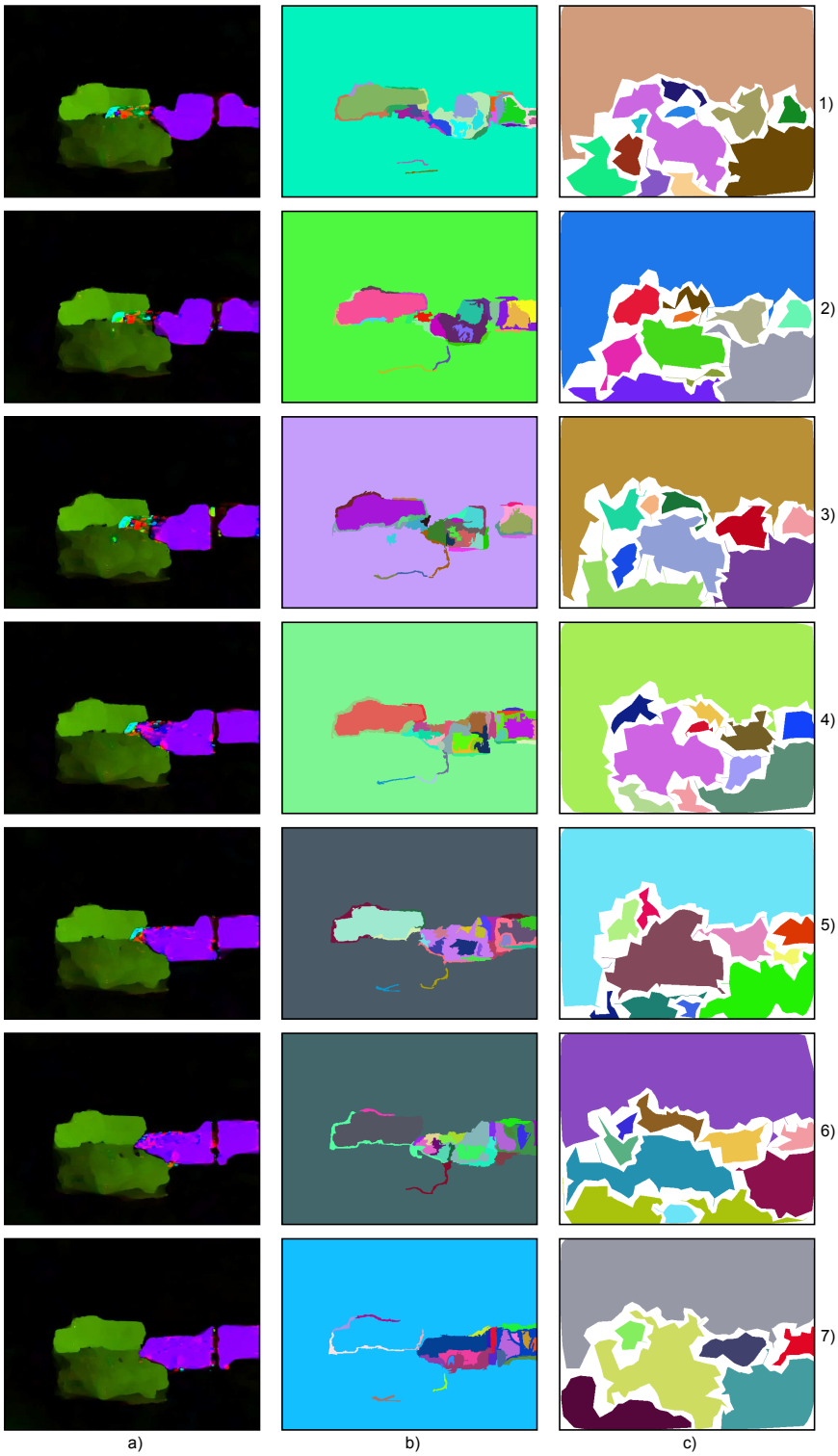

Figure 4.16: Segmentation results of dense and sparse optical flow images. Edge weight was calculated as exponential function and the angle was considered. a) Computed dense optical flow images from the dataset *Dumptruck* for the input images in fig. 4.2. b) Segmented dense optical flow after using images from a) as input. c) Segmented sparse optical flow after converting images from fig. 4.2 into grey-scale images and computing the sparse optical flow used as input for the segmentation.

Qualitative comparison

The following table shows the results of the qualitative comparison between the dense optical flow and sparse optical flow images respectively and the computed dense optical flow input images. The first column of the table contains the items that were checked while the second and third column contain the comparison results between the output images in fig. 4.16 b) and c) respectively and the input image in fig. 4.16 a).

Item of comparison	Segm. dense OF	Segm. sparse OF
All components exist	0/7	5/7
1 component lost	7/7	2/7
2 components merged into 1	0/7	5/7
1 components split in several parts	7/7	7/7

For the comparison the splitting caused by the algorithm due to its limitation of the magnitude was ignored. It can be distinguished from the splitting caused by the set of segmentation parameters by its border. Components split by the workaround have frazzled boarders.

The segmentation results in fig. 4.16 b) can not detect the car in the front[13] while the output in c) can. An additional post-processes can improve the output in fig. 4.16 c) but it can not insert missing components.

Due to the small value of k the same effects concerning splitting of components occur, see sec. 4.5.2 for more details.

Nevertheless it appears that the segmentation of sparse optical flow images yields better results because all objects form the input have corresponding components in the output (although they are split into several components).

Summary

For datasets like this where objects move very slowly the calculation of the edge weights using the exponential function is not suited.

4.6.3 Dataset Box06

The segmentation inputs as well as the results are shown in fig. 4.17. The dense optical flow images can be seen in fig. 4.17 a). Fig. 4.17 b) shows the segmented dense optical flow and c) the segmented sparse optical flow. For the computation of the sparse optical flow the input images from fig. 4.3 were used.

Equivalent to sec. 4.5.3 for the generation of the segmentation results the direction of the motion vector was ignored. Otherwise the workaround would cause splitting of components that make the comparison difficult. This approach is valid as long as the same set of parameter is used for segmentation and there is only one motion direction of the objects.

Qualitative comparison

The results of the qualitative comparison can be seen in the next table. The first column contains the items that were checked. The second column contains the comparison results between the

[13] This was already mentioned in sec. 4.3.2.

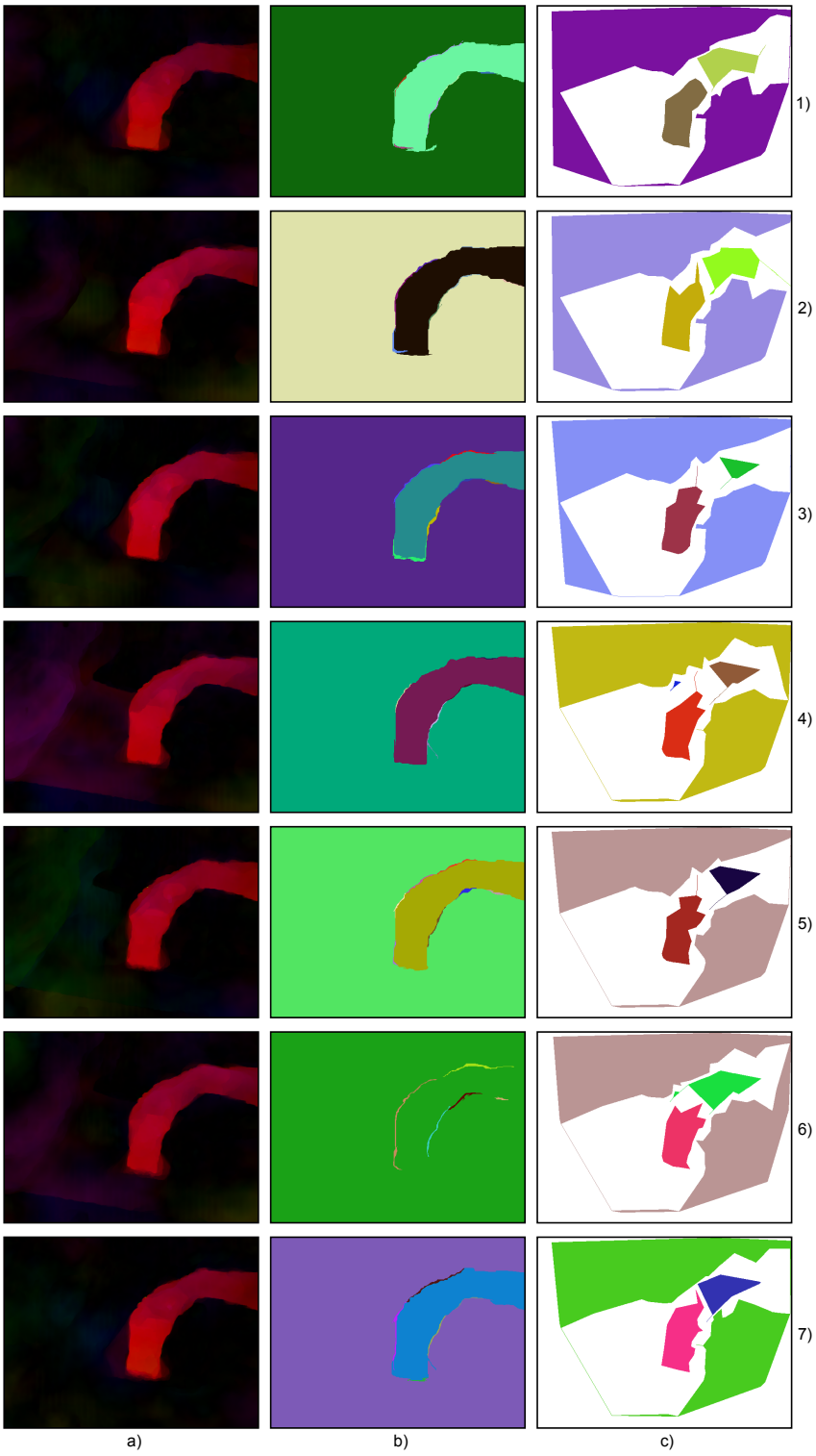

Figure 4.17: Segmentation results of dense and sparse optical flow images. Edge weight was calculated as exponential function and the angle was ignored. a) Computed dense optical flow images from the dataset *Box06* for the input images in fig. 4.3. b) Segmented dense optical flow after using images from a) as input. c) Segmented sparse optical flow after converting images from fig. 4.3 into grey-scale images and computing the sparse optical flow used as input for the segmentation.

output images in fig. 4.17 b) and the input images in fig. 4.17 a). Column three contains the results between images in fig. 4.17 c) and input images in fig. 4.3.

Note that the image sequence contains hundreds of color images and optical flow images. Seven dense optical flow images of them are shown in fig. 4.17 a).

Item of comparison	Segm. dense OF	Segm. sparse OF
All components exist	6/7	7/7
1 component lost	1/7	0/7
2 components merged into 1	6/7	0/7
1 components split in several parts	0/7	1/7

Using the dense optical flow for segmentation in one output frame the major component (box and hand clustered together) is lost, see fig. 4.17 b), frame 6. The segmentation output of the sparse optical flow contains all objects from the input. Similar to fig. 4.14 c) in fig. 4.17 c) the output contains also the objects from the input split into corresponding components.

Quantitative comparison

For this comparison the frame number two in fig. 4.17 was observed. The hand was not considered for this comparison. The box from the color image and the corresponding component from the dense and sparse optical flow image were cut out. The degree of matching was calculated using eq. 4.1.

The following table shows the results. The first column indicates the object that was selected. The second and third column show the degree of matching for the dense and sparse optical flow image respectively.

Object of comparison	Match (dense OF)	Match (sparse OF)
box=image	80.2%	88.6%

Summary

The extraction of corner points and thus the computation of the sparse optical flow is more exact than the computation of the dense optical flow. In the latter parts of the surrounding shadows are added to the box to form one object. The segmentation result of the sparse optical flow image yields the clearly better result. The ideal value of 100% is not reaches because of the approximation of the object border.

4.7 Color and optical flow images

This section describes the excursus concerning the segmentation of color and optical flow images[14]. The theory therefore was already mentioned in sec. 3.3. Because the major aim of this work is the analysis and evaluation of sparse and dense optical flow segmentation, the segmentation of combined images is briefly discussed and was only tested with the dataset Backyard.

[14]They are called *combined images* in the following.

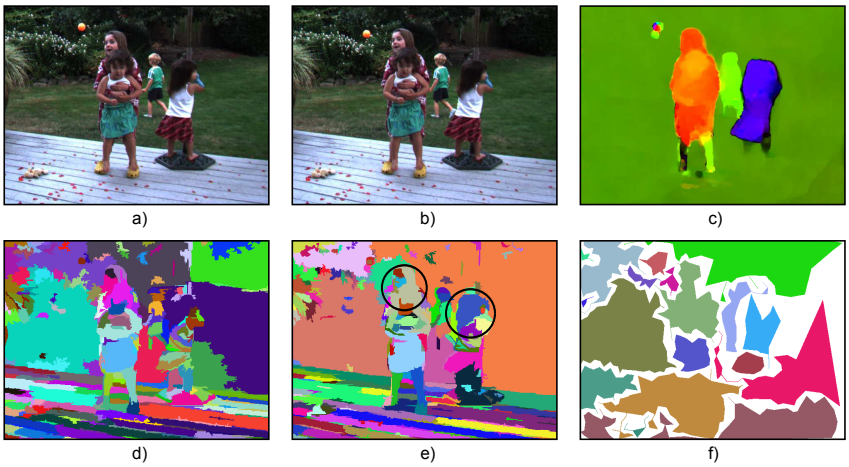

Figure 4.18: Segmentation results of combined images. a) and b) First two frames of dataset Backyard. c) Computed dense optical flow for images in a) and b). d) Segmented output (Euclidean distance, without angle) with image a) as input. e) Segmentation output with images in a) and c) as input. Differences to d) encircled. f) Segmentation output with images in a) and b) as input.

Fig. 4.18 shows all relevant input and output images. The input images in fig. 4.18 (a-b) were used for the calculation of the dense optical flow image in fig. 4.18 c) and for the sparse optical flow image. Fig. 4.18 d) shows the output if the color image in fig. 4.18 a) was used for segmentation. The segmentation result if the color image from fig. 4.18 a) and dense optical flow image from fig. 4.18 c) were used can be seen in fig. 4.18 e). The segmentation result of color and sparse optical flow image is shown in fig. 4.18 f). For the latter the color images from fig. 4.18 (a-b) were used as input.

Following segmentation parameters were used:

- Fig. 4.18 e): edge weight = Euclidean distance, $k = 200$, $min = 150$.
- Fig. 4.18 f): edge weight = Exponential function, angle of motion vector is considered, $k = 3$, $min = 100$, $\sigma_{rgb} = 44$, $\sigma_{uv} = 3$, $c_{angle} = 20°$.
- Fig. 4.18 g): edge weight = Exponential function, angle of motion vector is considered, $k = 3$, $min = 5$, $\sigma_{rgb} = 44$, $\sigma_{uv} = 3$, $c_{angle} = 20°$.

Compared to the segmented color image in fig. 4.18 d) the segmentation output for the combined image using the dense optical flow (fig. 4.18 e)) contains less details. Beside this the components are clustered different which achieves better the form of the real objects. This implies that adding the motion information improves the segmentation output of the color images alone. For example, the hairs of the left children are clustered together in fig. 4.18 e) because of similar color (and motion). The hairs of the right child is separated to one component. Parts of the background are clustered.

Compared to the segmented dense optical flow image in fig. 4.6 c) the segmentation result of the combined image using the dense optical flow in fig. 4.18 e) contains more details.

The degree of refinement and the count of components within a segmentation output depend on the application. For example, if segmentation is used for robot navigation the profile of the

moving objects is more important than the details itself. For this scenario the segmentation of optical flow images is better qualified than the segmentation of combined images.

Fig. 4.18 f) shows the segmentation result for a color and sparse optical flow image. Here the same conclusions as for fig. 4.18 e) can be drawn.

Chapter 5

Motion segmentation of multiple subsequent frames

The segmentation of single optical flow images is very detailed covered in ch. 4. This chapter highlights the excursus of the segmentation of several optical flow images, used for the detection of objects with consistent motion over several frames.

In the following the applied parameters, the algorithm and the results are discussed for the dataset *Box06*. It is shown that sparse optical flow images as well as dense optical flow images can be used as input.

Segmentation parameter and overlapping

For the segmentation of the optical flow images the same parameter sets as in sec. 4.3.3 and 4.4.3 were applied:

- Dense optical flow: edge weight = Exponential function, angle of motion vector is ignored, $k = 2$, $min = 100$, $\sigma_{uv} = 2$.
- Sparse optical flow: edge weight = Exponential function, angle of motion vector is ignored, $k = 3$, $min = 5$, $\sigma_{uv} = 2$.

Due to the hard threshold for the limitation of the motion vector's magnitude and the special image sequence the angle is ignored. See sec. 4.5.3 for further details.

For the comparison of the components in the subsequent segmentation results an overlapping by 75% was requested.

Algorithm

First a given set of input images is segmented and the components in their results are saved. The components from the first segmentation output are used for initialization of the reference image at the beginning. Thereafter a comparison between the objects from the adjacent segmentation output and the reference image is drawn. The overlapping for each component is calculated and if it is greater than or equal to the threshold this component is used for initialization of the new reference frame used for the next comparison. In this manner all segmentation results are checked.

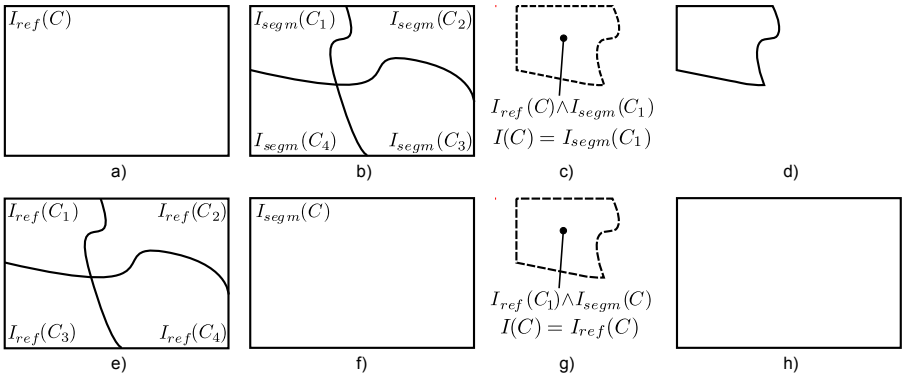

Figure 5.1: Example to demonstrate the robust handling of splitting. a) One component in the reference image. b) Corresponding component split into four parts (segmented optical flow image). c) Pixel-wise conjunction of two components, $I_{ref}(C) \wedge I_{segm}(C_1)$. d) Resulting component for the new reference image. e) Four components in the reference image. f) Corresponding component in the segmented optical flow image. g) same to c). h) same to d).

The formula for the overlapping can be expressed as

$$overlapping = \frac{I_{ref}(C_i) \wedge I_{segm}(C_i)}{I(C_i)} \cdot 100\%, \tag{5.1}$$

where $I_{ref}(C_i)$ is a binary image that contains one component from the reference image, $I_{segm}(C_i)$ is a binary image that contains the component from the segmented optical flow image and $I(C_i)$ is also a binary image that contains the smaller component of that two. In the binary images the pixels forming the component have the value 1 (other pixels have zero value). The logic operation $\wedge$ (conjunction) is applied on every pixel in the images and the result value is 1 if both pixels have the value 1, otherweise zero. The number of pixels with the value 1 within the images are counted and the ratio is computed.

Fig. 5.1 illustrates the overlapping formula. Fig. 5.1 a) shows the component of the reference image, fig. 5.1 b) shows the corresponding, split component in the segmented optical flow image, fig. 5.1 c) shows the pixel-wise conjunction of the components, $I_{ref}(C) \wedge I_{segm}(C_1)$ and fig. 5.1 d) shows the component that is used for initialization of the new reference image. The sub figures in 5.1 (e-h) show the example if a split component in the reference image is merged together in the segmented flow image.

The algorithm is based on following assumptions:

- The position of objects changes very slow.
- If the corresponding component from the reference image is split into several parts in the segmented flow image (fig. 5.1 (a-d)) it is assumed that this parts belong together.
- Split components in the reference image belong together if they can be represented as one component in the segmented flow image (fig. 5.1 (e-h)).
- Due to robust handling of component splitting, a component get only lost if the overlapping condition is not satisfied.

Results

The results of segmentation over several frames (max. 15 out of 30) are shown in fig. 5.2. Fig. 5.2 a) and c) show the results for segmented dense optical flow images and b) and d) show the results for segmented sparse optical flow images. In fig. 5.2 a1) two dense optical flow images, in fig. 5.2 a2) three dense optical flow images were segmented and so on. If fig. 5.2 a7) is reached it continues with c1) where 9 dense flow images were segmented. Fig. 5.2 (a1-a7) and (c1-c7) are defined similar for sparse optical flow images.

Note the results in fig. 5.2 a5) and a6), for example. For some reasons the moving objects (box and hand) get lost in fig. 5.2 a5) (it is clustered with the background), that represents the reference image for the next comparison. Now it is the same situation as in the example in fig. 5.1 (a-b). The component, lost in fig. 5.2 a5) is back in fig. 5.2 a6).

In the results for the sparse optical flow in fig. 5.2 b) and d) the box can be seen quite good. Also the hand can be recognized even if the form is (because of missing corner points) totally different.

Following conclusions can be made for the resulting output:

- The result only contains components that were already available in the first segmentation output.
- Components that appear at a later segmentation output do not have a corresponding component in the reference image. Thus the calculated overlapping is less than the threshold and the component gets lost.
- Uncorrelated components in the segmentation results, that are generated by mistake, also get lost.
- The result only contains components with consistent motion over all input images.

Figure 5.2: Results of multiple frames segmentation. a) and c) Subsequent results if dense optical flow images are used as input. b) and d) Subsequent results if sparse optical flow images are used as input. a1) and b1) Result after 2 flow images. a2) and b2) Result after 3 flow images and so on. c1) and d1) Result after 9 flow images, c2) and d2) Result after 10 flow images and so on.

Chapter 6

Discussion

In this chapter some practical issues concerning the project are discussed. Sec. 6.1 covers the segmentation efficiency of sparse and dense optical flow images as well as the achievable segmentation accuracy relating to object borders. Also the two different implementations for the edge weight formula (Euclidean distance and exponential function) are analyzed. Possible improvements for the segmentation algorithm and a short outlook are given in sec. 6.2.

6.1 Segmentation efficiency

The segmentation efficiency can be determined from different point of views. The following shows the analysis of the execution time for the computation of the optical flow as well as the segmentation. Furthermore the accuracy of the object borders in the segmentation results and the performance using the different edge weight formulas are covered.

Execution time

The execution time for the segmentation of a dense optical flow image with a resolution of 640x480 pixels and 1.220.972 edges is in the range of approximately $1s$[1]. The segmentation of a sparse optical flow image with circa 3.000 edges takes approximately $2ms$. From the values can be seen that the execution time for segmentation scales linearly with the number of edges.

The computation of a dense optical flow image from two subsequent grey-scale images with the same resolution as above takes about $0.12s$[2] and for a sparse optical flow image with circa 1000 corner points takes approximately $0.04s$. Here, the computation time of the optical flow does not scale linearly with the number of pixels. The computation of the dense optical flow image is much faster than of the sparse optical flow image. This is caused by the different implementations of the flow calculation algorithm. The dense optical flow computation is a graphic processing unit (GPU) accelerated implementation [ZPB07]. This means that the computationally intensive calculation is done by the graphics card that is more efficient than the central processing unit (CPUs) of a computer. On the other hand, the computation of sparse optical flow images is coded in C++ and calculated by the CPU without any timing optimization.

[1] Hardware configuration: Intel(R) Core(TM)2 Duo CPU, 2.66GHz; 2GB RAM; operating system Linux; nVidia GeForce 9800GT.

[2] In [ZPB07] the computation time in $[Frames/s]$ are given for different image resolutions. The linear relationship was used to calculate the computation time for an image with a resolution of 640x480 pixels.

Accuracy

From the results in ch. 4 can be seen that both optical flow segmentation methods have their advantages and shortcomings at certain circumstances.

Segmentation of dense optical flow images:

+ Object borders can be segmented with high accuracy if the quality of the images and the frame rate[3] is accordingly high.

- If the conditions concerning high image quality and frame rate are not satisfied the computation of the dense flow is erroneous and thus the segmentation becomes impossible.

- The computation of a motion vector for every pixel in the image is computationally intensive.

Segmentation of sparse optical flow images:

- Because of the limited number of corner points by which the object border is approximated the outline of the component is somewhat different from the object itself.

+ The extraction of corner points is quite robust even in images with low quality, thus the image quality plays a minor role here. This means, that this segmentation delivers results even if the segmentation of dense optical flow images fails.

+ Motion vectors are only computed for certain pixels in the image which is resource-saving[4].

Depending on which objective is most important, object border accuracy, computational efficiency, or desired image quality, one of the segmentation algorithms mentioned above (dense optical flow or sparse optical flow) is preferable. For example, if only the detection of a moving object is desired, the segmentation of sparse optical flow images is adequate because the accurate position of object borders is not of interest. However, if object borders have to be precisely segmented most likely dense optical flow images have to be used.

Edge weight formulas

For the computation of the edge weights, used by the segmentation algorithm to build a graph, two different formulas were implemented and tested. The theory was introduced in sec. 3.2.3 and the results were shown in ch. 4.

The advantages and disadvantages of the edge weight calculated as exponential function over the Euclidean distance is shown in following list:

- A small change of the value k, used for the calculation of the threshold, causes totally different segmentation results. This means that the variation of k for the segmentation using the Euclidean distance does not have the same strong effect as for the segmentation using the exponential function.

[3] In general a frame rate of $25Hz$ is used.

[4] Therefore it has to be assumed that the implementation of the sparse optical flow computation is also carried by the GPU (and not the CPU). Otherwise a serious comparison between sparse and dense optical flow computation can not be drawn.

+ Components in the segmentation output are more likely split into several parts. Thus, the segmentation result contains more details that can easily be merged in a post-process that is needed anyway. Note that it would be much more difficult to split components in a post-process that were merged by mistake.

Depending on the motion of the image sequence and the quality of the frames one of the two edge weight formulas has to be preferred. This is also mirrored in the tables of the qualitative and quantitative comparisons in ch. 4.

6.2 Outlook

The implemented segmentation algorithm and its results shown in the previous chapters are satisfying but there is still some room for improvements.

Limitation of the motion vector's magnitude

For image sequences with a static camera the magnitude of the motion vectors in regions of non-moving objects is not exactly zero, but very small and has an arbitrary angle. The implementation in this work uses a fixed threshold to re-set the magnitude which is a disadvantage (see sec. 4.3.2). The overall result can be improved by using a two-stage implementation. In the first step a certain threshold determined with the optical flow computation accuracy is applied to the magnitude. Those pixels smaller than this threshold are re-set with the calculated mean value of motion vectors within a defined neighboring surrounding. This makes sure that outliers within a component caused by the thresholding are eliminated. In a second thresholding step the angle of pixels of low magnitude is set to a constant value in order to make sure that they can be segmented as a continuous component. Additionally, the magnitude of those pixels is set to zero.

Identical parameter sets

A closer look on the parameter sets used for the segmentation of the different image sequences in ch. 4 shows that they are different for each dataset but quite similar within a dataset for segmented dense and sparse optical flow images, respectively. The following table illustrated this for the dataset *Backyard* and *Box06*[5]:

Backyard	dense OF, euc	dense OF, exp	sparse OF, euc	sparse OF, exp
k	110	3	90	3
min	50	50	5	5
σ_{uv}	-	3	-	3

Box06	dense OF, euc	dense OF, exp.	sparse OF, euc	sparse OF, exp
k	60	2	10	3
min	100	100	5	5
σ_{uv}	-	2	-	2

[5]The dataset *Dumptruck* was not considered, because the segmentation of dense optical flow images using the exponential function for edge weight calculation failed. Beside this, the dataset *Box06* is from the same style (static camera, moving objects).

As far as the direction of the motion vector was considered, the tolerance angle was set to $c_{angle} = 20°$.

The differences between the parameters of sparse and dense optical flow segmentation within a dataset is caused by the different number of edges. For example, the value *min* that is used by the post-process to remove small components, has to be smaller in case of sparse optical flow images where only a few pixels (corner points) form a component.

Within one dataset the parameters for the segmentation of the dense optical flow images are the same for both edge weight formulas except for the value k. The same holds true for sparse optical flow images. The different values for k are caused by the different output range of the weight functions. As mentioned in sec. 3.2.3 the maximum value for the edge weight between two totally dissimilar pixels calculated as Euclidean distance is $w_{euclid,max} = 282$, while it is $w_{exp,max} = 1$ if the weight is calculated using the exponential function. Similar parameter sets are expected if the output range of the weight functions would be normalized.

Bibliography

[ADK99] G. Aubert, R. Deriche, and P. Kornprobst. Computing optical flow via variational techniques. *SIAM Journal on Applied Mathematics*, 60(1):156–182, 1999.

[BA93] M. J. Black and P. Anandan. A framework for the robust estimation of optical flow. *International Conference on Computer Vision*, pages 231–236, 1993.

[BK08] G. Bradski and A. Kaehler. *Learning OpenCV: Computer Vision with the OpenCV Library*. O'Reilly, 2008.

[Bor04] S. Borman. The expectation maximization algorithm - a short tutorial. July, 2004.

[Bou00] J.-Y. Bouguet. Pyramidal implementation of the lucas kanade feature tracker. 2000.

[BVZ99] Y. Boykov, O. Veksler, and R. Zabihn. Fast approximate energy minimization via graph cuts. *International Conference on Computer Vision*, pages 377–384, 1999.

[CC06] F.-H. Cheng and Y.-L. Chen. Real time multiple objects tracking and identication based on discrete wavelet transform. *Pattern Recognition*, 39(6):1126–1139, 2006.

[CFM07] A. Colombari, A. Fusiello, and V. Murino. Segmentation and tracking of multiple video objects. *Pattern Recognition*, 40(4):1307–1317, 2007.

[CK98] J. P. Costeira and T. Kanade. A multibody factorization method for independently moving objects. *International Journal of Computer Vision*, 29(3):159–179, 1998.

[CS05] D. Cremers and S. Soatto. Motion competition: A variational approach to piecewise parametric motion segmentation. *International Journal of Computer Vision*, 62(3):249–265, May, 2005.

[CSE05] A. Cavallaro, O. Steiger, and T. Ebrahimi. Tracking video objects in cluttered background. *IEEE Transactions on Circuits and Systems for Video Technology*, 15(4):575–584, 2005.

[DW01] Song Wang Department and Song Wang. Image segmentation with minimum mean cut. In *IEEE Computer Society*, pages 517–524, 2001.

[FH04] P. F. Felzenzswalb and D. P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2), 2004.

[GB98] P. L. George and H. Borouchaki. *Delaunay Triangulation and Meshing*. Hermes, 1998.

[HB04] J. Hawkins and S. Blakslee. *On intelligence*. Times Books, Henry Holt and Company, LLC, 1st edition, 2004.

[HHD00] I. Haritaoglu, D. Harwood, and L. S. Davis. W4: Real-time surveillance of people and their activitie. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 22(8):809–830, August, 2000.

[HS80] B. K. P. Horn and B. G. Schunck. Determining optical flow. *Artificial Intelligence*, 17:185–203, March, 1980.

[JSL+07] C. Julia, A. Sappa, F. Lumbreras, J. Serrat, and A. Lopez. Motion segmentation from feature trajectories with missing data. *Iberian Conference on Pattern Recognition and Image Analysis*, pages I:483–490, 2007.

[KLGW98] M. Kong, J.-P. Leduc, B. Ghosh, and V. Wickerhauser. Spatio-temporal continuous wavelet transforms for motion-based segmentation in real image sequences. *Proceedings of the International Conference on Image Processing*, 2:662–666, Octobr, 1998.

[KTZ08] M. P. Kumar, P. H. Torr, and A. Zisserman. Learning layered motion segmentations of video. *International Journal of Computer Vision*, 76(3):301–319, 2008.

[LK81] B. D. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. *International Joint Conference on Artificial Intelligence*, pages 674–679, 1981.

[LYY07] R. Li, S. Yu, and X. Yang. Efcient spatio-temporal segmentation for extracting moving objects in video sequences. *IEEE Transactions on Consumer Electronics*, 53(3):1161–1167, August, 2007.

[NE86] H.-H. Nagel and W. Enkelmann. An investigation of smoothness constraints for the estimation of displacement vector fields from image sequences. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 8:565–593, 1986.

[OAT+07] E. Ohashi, T. Aikoand, T. Tsuji, H. Nishi, and K. Ohnishi. Collision avoidance method of humanoid robot with arm force. *IEEE Transaction on Industrial Electronics*, 54(3):1632–1641, June, 2007.

[PBB+06] N. Papenberg, A. Bruhn, T. Brox, S. Didas, and J. Weickert. Highly accurate optic flow computation with theoretically justified warping. *International Journal of Computer Vision*, 67(2):141–158, April, 2006.

[RH01] C. Rasmussen and G. D. Hager. Probabilistic data association methods for tracking complex visual objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(6):560–576, 2001.

[SGHG08] R. Stolkin, A. Greig, M. Hodgetts, and J. Gilby. An em/e-mrf algorithm for adaptive model based tracking in extremely poor visibility. *Image and Vision Computing*, 26(4):480–495, 2008.

[SGK00] H. S. Sawhney, Y. Guo, and R. Kumar. Independent motion detection in 3d scenes. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 22(10):1191–1199, October, 2000.

[SHB99] M. Sonka, V. Hlavac, and R. Boyle. *Image Processing, Analysis, and Machine Vision*. PWS, 2nd edition, 1999.

[SM00] J. Shi and J. Malik. Nomalized cuts and image segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(8), 2000.

[SPV09] M. J. Schlemmer, J. Prankl, and M. Vincze. Vision for situated robot companions - fusing top-down knowledge and bottom-up data. 2009.

[SZHL07] H. Shen, L. Zhang, B. Huang, and P. Li. A map approach for joint motion estimation, segmentation, and super resolution. *IEEE Transactions on Image Processing*, 16(2):479–490, 2007.

[TK91] C. Tomasi and T. Kanade. Detection and tracking of point features. Technical report, Carnegie Mellon University, Technical Report CMU-CS-91-132, April, 1991.

[TK92] C. Tomasi and T. Kanade. Shape and motion from image streams under orthography: a factorization method. *International Journal of Computer Vision*, 9(2):137–154, 1992.

[Twe09] David S. Tweed. *Motion Segmentation Across Image Sequences*. PhD thesis, Department of Computer Science, University of Bristol, April, 2009.

[VJS05] P. A. Viola, M. J. Jones, and D. Snow. Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision*, 63(2):153–161, July, 2005.

[VTY07] N. Vaswani, A. Tannenbaum, and A. Yezzi. Tracking deforming objects using particle ltering for geometric active contours. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(8):1470–1475, 2007.

[WB02] J. Weickert and T. Brox. Diffusion and regularization of vector- and matrix-valued images. *Inverse Problems, Image Analysis ans Medical Imaging. Contemporary mathematics*, 313:251–268, 2002.

[Wis97] L. Wiskott. Segmentation from motion: Combining gabor- and mallat-wavelets to overcome aperture and correspondence problem. *Proceedings of the 7th International Conference on Computer Analysis of Images and Patterns*, 1997.

[YP06] J. Yan and M. Pollefeys. A general framework for motion segmentation: Independent, articulated, rigid, non-rigid, degenerate and non-degenerate. *European Conference on Computer Vision*, pages IV:94–106, 2006.

[ZLS08] L. Zappella, X. Llado, and J. Salvi. Motion segmentation: A review. 2008.

[ZPB07] C. Zach, T. Pock, and H. Bischof. A duality based approach for realtime tv-l1 optical flow. In *Pattern Recognition (Proc. DAGM)*, pages 214–223, Heidelberg, Germany, 2007.

[ZSWL07] J. Zhang, F. Shi, J. Wang, and Y. Liu. 3d motion segmentation from straight-line optical ow. *Multimedia Content Analysis and Mining*, pages 85–94, 2007.